Verbunddokumente als Nutzeroberfläche von Software für die Tragwerksplanung

Dissertation

zur Erlangung des akademischen Grades

Doktor-Ingenieur

an der Fakultät Bauingenieurwesen
der Bauhaus-Universität Weimar

Vorgelegt von Dipl.-Ing. Daniel Bittrich aus Chemnitz

Weimar

Gutachter:
1. Prof. Dr.-Ing. Karl Beucke, Bauhaus-Universität Weimar
2. Prof. Dr.-Ing. Stefan Holzer, Universität Stuttgart
3. Prof. Dr.-Ing. habil. Erich Raue, Bauhaus-Universität Weimar

Tag der Disputation: 14. Dezember 2001

Vorwort

Die vorliegende Arbeit entstand in den Jahren 1997 bis 2001 im Rahmen meines Promotionsstudiums als Stipendiat des Freistaates Thüringen am Lehrstuhl Informatik im Bauwesen der Bauhaus-Universität Weimar.

An erster Stelle möchte ich Prof. Dr. Karl Beucke erwähnen, der seit seiner Berufung zum Professor für Informatik im Bauwesen meinen Weg maßgeblich beeinflusst hat. Durch sein Engagement in Forschung und Lehre wurde ich in meinem Interesse an der Bauinformatik bestärkt und habe mich schließlich für eine mehrjährige wissenschaftliche Arbeit entschieden. Für die zahlreichen persönlichen Gespräche, die individuelle Betreuung und schließlich für die Übernahme der Begutachtung bin ich ihm sehr dankbar.

Ebenso möchte ich mich bei allen meinen Kollegen am Lehrstuhl bedanken, insbesondere bei Dr. Wolfgang Huhnt und Berthold Firmenich für die zahlreichen Diskussionen und Anregungen.

Prof. Dr. Stefan Holzer hat mit seiner Veröffentlichung "Gestaltung ingenieurgemäßer Statiksoftware" wesentlich zu meiner Motivation für diese Arbeit beigetragen und schließlich die Begutachtung übernommen. Dafür möchte ich mich bei ihm bedanken.

Bei Prof. Dr. Erich Raue möchte ich mich ebenfalls für die Übernahme der Begutachtung bedanken und außerdem für die Unterstützung bei der Bewerbung für mein Promotionsstipendium.

Schließlich möchte ich mich auch bei meiner Familie und meinen Freunden für Unterstützung und Motivation bedanken. Mein ganz spezieller Dank gilt meiner Freundin Mirja Pötschke, die stets Zeit und Verständnis für meine Arbeit aufgebracht hat und mir ein tiefgründiger Gesprächspartner und sorgfältiger Korrekturleser gewesen ist.

Weimar, den 16. Mai 2001

Inhaltsverzeichnis

Kapitel 1
Einführung

Die Entwicklung von Modellen, Verfahren und Werkzeugen für die Tragwerksplanung gehört zu den klassischen Aufgaben der Bauinformatik. Der Schwerpunkt von Forschung und Entwicklung auf dem Gebiet der Tragwerksplanungs-Software lag lange Zeit auf der kontinuierlichen Erweiterung des Funktionsumfangs, so dass mittlerweile für nahezu alle Problemstellungen des Tragwerksplaners leistungsfähige Software-Lösungen zur Verfügung stehen. In der Folge ist es notwendig, den bereits vorhandenen Funktionsumfang einem möglichst breiten Anwenderkreis durch ingenieurgemäß gestaltete Nutzeroberflächen besser zugänglich zu machen, so dass ein möglichst effizientes und fehlerarmes Arbeiten ermöglicht wird.

Obwohl der Stellenwert ingenieurgemäß gestalteter Nutzeroberflächen seit längerem bekannt ist, wurde dies bislang bei der Entwicklung von Tragwerksplanungs-Software nur selten angemessen berücksichtigt. Eine Ursache dafür ist der im Vergleich zur Entwicklung der Kernfunktionalität unverhältnismäßig hohe Aufwand zur Erstellung ingenieurgemäßer Nutzeroberflächen. In dieser Arbeit wird ein Konzept vorgestellt, das zur Überwindung dieses sowohl für Tragwerksplaner als auch für Software-Entwickler unbefriedigenden Zustandes beitragen kann.

1.1 Problembeschreibung

Der Markt für Tragwerksplanungs-Software ist geprägt durch einige allgemeine, zumeist auf Finiten Elementen basierende Systeme großer Komplexität und eine Vielzahl von kleinen und mittleren Anwendungen zur Analyse und konstruktiven Durchbildung spezieller Bauteiltypen (Platten, Unterzüge, Stützen, Dächer, Fundamente). Die Nutzeroberflächen sind in vielen Fällen aus entwicklungsgeschichtlichen Gründen nur bedingt konform mit den aktuellen Gestaltungsrichtlinien. Nur vereinzelt und nicht mit voller Konsequenz werden innovative Konzepte verfolgt.

Bei der gegenwärtig typischen Arbeitsweise zerlegt der Ingenieur das Tragwerk in überschaubare Bauteile. Für diese sogenannten Positionen werden die Schnittgrößen ermittelt und damit eine Bemessung durchgeführt sowie deren Tragsicherheit und Gebrauchstauglichkeit nachgewiesen. Anschließend werden die Ergebnisse im Tragwerksbericht dokumentiert. Im Regelfall wird für jeden Arbeitsschritt ein eigenständiges Programm verwendet. Somit sind zur Bearbeitung eines Bauteils bis zu drei verschiedene Anwendungen erforderlich, von denen jede über eine eigene Nutzeroberfläche und eine eigene Datenhaltung verfügt. Infolge der häufig inkonsistenten Nutzeroberflächen und unbefriedigenden Interaktionsmöglichkeiten zwischen den beteiligten Anwendungen werden Tragwerksberichte in vielen Fällen nach wie vor handschriftlich oder mit allgemeiner Textverarbeitungs-Software erstellt und durch separate Ausdrucke der verwendeten Tragwerksplanungsprogramme ergänzt.

Eine Anpassung von Tragwerksplanungs-Software an die individuellen Wünsche des Tragwerksplaners ist gegenwärtig gar nicht oder nur begrenzt über prozedurale Makrosprachen möglich. Weiterhin finden sich in der Literatur bislang keine Hinweise darauf, wie Standard-Software an die Erfordernisse der Tragwerksplanung angepasst werden kann.

In der Software-Entwicklung werden Wartung und Weiterentwicklung der kontinuierlich gewachsenen Systeme immer aufwändiger. Unzureichende Möglichkeiten zur Wiederverwendung von Programm-Code, zur Einbindung von Standard-Software und zur Erweiterung bestehender Systeme durch externe Entwickler führen zu kostspieligen Mehrfachentwicklungen. Ein viel zitiertes Beispiel stellt die Textverarbeitungsfunktionalität dar, die immer wieder durch eigene Implementierungen realisiert wird. Bei den für das Bauwesen typischen, vergleichsweise geringen Stückzahlen wirken sich die genannten Punkte besonders negativ auf das Preis-Leistungs-Verhältnis von Tragwerksplanungs-Software aus.

Zusammenfassend lässt sich feststellen, dass Tragwerksplanungs-Software zwar im Allgemeinen den fachlichen Erwartungen der Anwender entspricht, aber im Hinblick auf die Gestaltung der Nutzeroberflächen und der Interaktionsmöglichkeiten mit anderen Applikationen häufig veraltet ist. Dafür werden zwei Ursachen identifiziert:

1. Traditionell wurde ein großer Teil der in der Tragwerksplanung eingesetzten Software von Bauingenieuren selbst entwickelt. Mit dem Übergang von alphanumerischen zu grafischen Nutzeroberflächen und dem damit einhergehenden Paradigmenwechsel von der prozeduralen zur objektorientierten Programmierung wurde in der Software-Entwicklung eine Stufe der Komplexität erreicht, deren Beherrschung eine fundierte theoretische Ausbildung und entsprechende praktische Erfahrungen in der Software-Entwicklung erfordert. Über diese software-technische Professionalität verfügen Bauingenieure, die hauptsächlich in ihrem angestammten Metier arbeiten und sich nur nebenbei mit den aktuellen Entwicklungen der Informatik beschäftigen, nur noch in Ausnahmefällen. Damit praktisch tätige Bauingenieure wieder aktiv Einfluss auf die Entwicklung von Tragwerksplanungs-Software nehmen können, muss die Komplexität der Software-Entwicklung reduziert werden.

2. Einhergehend mit dem tendenziell sinkenden Anteil der Bauwirtschaft an der Gesamtwirtschaft ergeben sich auch für die Hersteller von Tragwerksplanungs-Software nur noch unterdurchschnittliche Wachstumsmöglichkeiten. In dieser Situation herrscht unter den Software-Herstellern rigoroser Verdrängungswettbewerb. Es besteht kaum Aussicht auf angemessene Gewinne, woraus sich wiederum eine geringe Investitionsneigung ergibt. Allein mit den bisher im Hinblick auf zu erwartende Skalen- und Synergieeffekte ergriffenen strukturellen Maßnahmen – wie Übernahmen und Zusammenschlüsse von Unternehmen – kann dieser Zustand nicht überwunden werden. Infolgedessen müssen auch alle software-seitigen Möglichkeiten ausgenutzt werden, die dazu beitragen können, den Ressourceneinsatz zu senken beziehungsweise den Wünschen der Anwender besser zu entsprechen und dadurch die Rentabilität wieder auf das erforderliche Mindestmaß zu steigern.

1.2 Zielstellung

Das vordergründige Ziel dieser Arbeit ist die Entwicklung von Konzepten, die zu einer intuitiv zu bedienenden Tragwerksplanungs-Software führen und dadurch zu deren effizienter und fehlerarmer Anwendung beitragen können. Dazu ist zunächst zu klären, durch welche Merkmale sich eine ingenieurgemäße Nutzeroberfläche auszeichnet und inwieweit von Standard-Software bekannte Benutzungskonzepte übernommen werden können. Neben den Möglichkeiten zur Vereinfachung der Benutzung von einzelnen Anwendungen muss auch untersucht werden, wie der Informationsfluss zwischen den beteiligten Anwendungen – zumindest für den Prozess der Erstellung des Tragwerksberichtes – optimiert werden kann. Besonderer Wert ist auf eine Trennung der allgemeinen Entwicklungsaufgaben und der Entwicklung fachspezifischer Inhalte zu legen, so dass eine zweckmäßige Aufgabenteilung zwischen Ingenieuren und Software-Entwicklern ermöglicht wird. Um die praktische Anwendung der entwickelten Konzepte sicherzustellen, muss eine Anpassbarkeit der Tragwerksplanungs-Software durch den Anwender an seine individuellen Bedürfnisse gegeben sein.

Obwohl die theoretischen Grundlagen der Gestaltung von Tragwerksplanungs-Software bereits umfassend untersucht sind, wurden die entwickelten Konzepte häufig nur in Ansätzen oder gar nicht bei der Entwicklung von kommerziellen Produkten aufgegriffen. Eine genauere Analyse der Ursachen für diesen unbefriedigenden Zustand ist erforderlich. Vorab wird jedoch die These aufgestellt, dass in vielen Fällen zunächst zu wenig Rücksicht auf wirtschaftliche Gesichtspunkte genommen wurde und sich deshalb später der Realisierungsaufwand als unverhältnismäßig hoch erwiesen hat.

Daraus ergibt sich als weiteres Ziel, dass die entwickelten Konzepte unter möglichst geringem Ressourceneinsatz für kommerzielle Anwendungen genutzt werden können. Ein vielversprechender Ansatz zur Begrenzung des Entwicklungsaufwandes liegt in einer möglichst umfassenden Wiederverwendung bereits implementierter Funktionalität. Neben der Wiederverwendung von Funktionalität, die in einer Organisation intern zur Verfügung steht, sind auch Möglichkeiten zu prüfen, inwieweit auf Implementierungen externer Entwickler, insbesondere in Form von Standard-Software, zurückgegriffen werden kann. Umgekehrt erscheint es auch zweckmäßig, selbst entwickelte Software externen Entwicklern zur Verfügung zu stellen.

Anhand einer Pilotimplementierung ist zu zeigen, wie sich aus der Sicht der Software-Hersteller Konzepte für ingenieurgemäß gestaltete Tragwerksplanungs-Software mit vertretbarem Aufwand umsetzen lassen. Der Schwerpunkt soll dabei ausdrücklich nicht auf seltenen Spezialaufgaben liegen, sondern auf der Unterstützung immer wiederkehrender Tätigkeiten, die den überwiegenden Anteil des Ingenieuralltags ausmachen. Mit einem Beispiel aus der bauteilorientierten Tragwerksplanung ist zu demonstrieren, dass der implementierte Prototyp aus der Sicht der Tragwerksplaner eine spürbare Arbeitserleichterung darstellt.

1.3 Vorgehensweise

In Kapitel 2 wird als Ausgangspunkt der Arbeit der aktuelle Stand der Technik von Tragwerksplanungs-Software dargelegt. Dazu werden exemplarisch vier Vorarbeiten vorgestellt. Es wird diskutiert, welcher Beitrag durch diese Arbeiten zur ingenieurgemäßen Gestaltung von Tragwerksplanungs-Software bereits geliefert wurde, aber auch an welchen Stellen noch Bedarf für weitere Forschungsarbeiten besteht. Danach wird die Zielstellung für die eigene Arbeit präzisiert.

Um die spätere Umsetzbarkeit in kommerzielle Produkte zu erleichtern, müssen bei der Entwicklung von Konzepten zur ingenieurgemäßen Gestaltung von Tragwerksplanungs-Software die aktuellen Entwicklungen der Software-Technik von Beginn an berücksichtigt werden. Aus diesem Grund werden im Kapitel 3 in abstrakter Form die Grundlagen der komponentenorientierten Software-Technik diskutiert und in Kapitel 4 die konkreten Basistechnologien gegenübergestellt.

Gegenstand von Kapitel 5 ist die Formulierung von Zielen für den Entwurf von Software-Architekturen komponentenbasierter Systeme und die Entwicklung von Strategien für deren Erreichung.

In Kapitel 6 werden zunächst die Arbeitsabläufe und der Informationsfluss in der Tragwerksplanung analysiert. Danach werden die Merkmale von ergonomisch gestalteten Nutzeroberflächen benannt und verschiedene Konzepte diskutiert, die aus der Sicht der Tragwerksplaner zu ingenieurgemäß gestalteter Tragwerksplanungs-Software beitragen können. Da die Erfüllung der im vorangegangenen Kapitel benannten Anforderungen der Tragwerksplaner einen sehr hohen Entwicklungsaufwand erfordert, werden diese in Kapitel 7 aus der Sicht der Software-Entwickler um eine tragfähige software-technische Konzeption ergänzt. Der Schwerpunkt liegt dabei auf der Begrenzung des Entwicklungsaufwandes.

In Kapitel 8 werden die Konzepte aus Kapitel 5 bis Kapitel 7 zu einem konkreten Entwurf einer Pilotimplementierung weiterentwickelt. Dabei werden die Vor- und Nachteile verschiedener Varianten gegenübergestellt.

In Kapitel 9 wird die Realisierung der Pilotimplementierung und in Kapitel 10 die Erprobung anhand eines konkreten Anwendungsbeispiels aus dem Massivbau detailliert dokumentiert.

Als Abschluss der Arbeit bildet Kapitel 11 mit einer Bewertung aus der Sicht der Tragwerksplaner und der Software-Entwickler sowie einem Ausblick auf die Möglichkeiten einer Weiterentwicklung.

Kapitel 2
Stand der Technik von Tragwerksplanungs-Software

Entsprechend der Zielstellung der Arbeit werden in diesem Kapitel Vorarbeiten analysiert, die wesentliche Beiträge zur Gestaltung der Interaktionen zwischen Nutzer und Software sowie zur Erstellung und Dokumentation des Tragwerksberichtes geliefert haben. Aus der in Frage kommenden Vielfalt werden exemplarisch die Arbeiten von (Kühnemann, 1992), (Molkenthin, 1994a), (Werkle et al., 1996) und (Hinz, 1998) ausgewählt, da diese eine umfassende Darstellung von Konzepten und deren Realisierung beinhalten. Dabei wird insbesondere untersucht, warum die vorgeschlagenen Konzepte nur in Ansätzen oder gar nicht bei der Entwicklung von kommerziellen Produkten aufgegriffen wurden. Bei der Bewertung der vorgestellten Ansätze ist es notwendig, diese im Hinblick auf die ingenieurgemäße Gestaltung und software-seitige Umsetzung zu diskutieren. Für eine umfassende Darstellung der dazu benötigten Grundlagen wird auf die folgenden Kapitel verwiesen.

2.1 Kühnemann, 1992

Eine der ersten Untersuchungen, wie der Tragwerksplaner bei der Durchführung von Tragwerksanalysen unterstützt werden kann, wurde von (Kühnemann, 1992) und (Kühnemann et al., 1996) durchgeführt und dokumentiert. Der Schwerpunkt der Untersuchung lag dabei auf der Erstellung der Dokumentation in Form eines Tragwerksberichtes. Dazu wurde eine als Statikeditor bezeichnete integrierte Arbeitsumgebung entwickelt, welche das Erstellen von formatierten Texten, die Darstellung von Formeln in mathematischer Notation sowie das Einbinden von Skizzen und Grafiken ermöglicht. Ein besonderes Funktionsmerkmal stellt der parametrisierte Formelsatz dar, der spätere Aktualisierungen unterstützt und somit das Arbeiten mit vorgefertigten Rechenblättern ermöglicht.

Zunächst wurde untersucht, inwieweit zur Realisierung des Statikeditors auf Funktionalität von Standard-Software (unter anderem Textverarbeitungs-, Tabellenkalkulations-, Mathematik- und CAD- Software) zurückgegriffen werden kann. Dabei wurde festgestellt, dass keine der untersuchten Programmgattungen ausreichend den Bedürfnissen des Tragwerksplaners entspricht. Zum Zeitpunkt der Untersuchung verfügte die Standard-Software entweder über gar keine oder nur über inadäquate Programmierschnittstellen. Auf diesem Weg konnten die als notwendig erachteten weitreichenden Anpassungen nicht vorgenommen werden. Alternativ hätten die Erweiterungen und die anschließende Integration mehrerer unabhängiger Anwendungen zu einem integrierten Statikeditor nur auf der Ebene des Quell-Codes stattfinden können. Da Standard-Software im Regelfall nicht im Quell-Code zugänglich gemacht wird, konnte lediglich auf die Funktionalität eines frei verfügbaren Texteditors zurückgegriffen werden. Die darüber hinaus benötigte Funktionalität wurde durch Eigenentwicklungen abgedeckt.

Nach einer unbefriedigenden Realisierung unter MS-DOS und AutoCAD wurde OS/2 als Entwicklungsplattform verwendet. Die Implementierung erfolgte prozedural in der Sprache C. Zur Datenhaltung wurde ein proprietäres Dateiformat definiert.

Insgesamt stellt die Arbeit einen richtungsweisenden Ansatz dar. Wesentliche Anforderungen an Software für die Tragwerksplanung, wie die Unterstützung der Dokumentation, wurden erkannt und in Angriff genommen. Bedingt durch den damaligen Stand der Technik konnten die Ideen aber aus heutiger Sicht nicht zufriedenstellend realisiert werden.

2.2 Molkenthin, 1994

In (Molkenthin, 1994a), (Molkenthin, 1994b), (Molkenthin, 1995), (Molkenthin, 1996a) und (Molkenthin, 1996b) wurde theoretisch nachgewiesen, dass sich Berechnung und Bemessung von Tragwerken sowie deren Dokumentation in eine einheitliche Arbeitsumgebung – den sogenannten Statischen Editor – integrieren lassen.

Zunächst wurden zwei grundlegende Alternativen diskutiert, um verschiedene Werkzeuge in eine einheitliche Umgebung zur durchgängigen rechnergestützten Arbeit in der Tragwerksplanung einzubinden:

- Integration über Schnittstellen
- Integration über ein zentrales Datenmodell

Der Integrationsansatz über Schnittstellen ermöglicht bekanntermaßen nur die Übertragung der Schnittmenge der von den beteiligten Systemen und der Schnittstelle darstellbaren Informationen. Weiterhin steigt der Integrationsaufwand mit der Anzahl der zu unterstützenden Produkte überproportional an. Aus diesen Überlegungen heraus wird der Integrationsansatz über ein zentrales Datenmodell gewählt. Da bislang noch kein allgemein akzeptiertes Datenmodell spezifiziert werden konnte, wurde ein proprietäres Datenmodell für die Tragwerksplanung definiert und völlig auf die Einbindung von bereits existierenden Anwendungen verzichtet.

Das Gesamtmodell des Statischen Berichts wird in die drei Teilmodelle Tragwerk, Darstellung und Medien gegliedert.

Das Tragwerksmodell wird durch die abstrakten Basisklassen Tragobjekt und Anschluss geprägt. Von der Basisklasse Tragobjekt werden konkrete Klassen für die üblichen Bauteile wie Unterzug, Rahmen und Fundament abgeleitet. Von der Basisklasse Anschluss müssen alle möglichen Anschlusstypen wie Unterzug/Rahmen oder Rahmen/Fundament spezialisiert werden. In diesen Klassen werden Regeln formuliert, wie beispielsweise Unterzüge ihre Auflagerkräfte auf Rahmen aufbringen. Bauteil- und Anschlusstypen, die nicht in diesem Modell enthalten sind, können keiner konkreten Klasse zugeordnet werden und somit auch nicht erfasst werden.

Das Darstellungsmodell ist definiert durch Klassen für Fließtext, Tabellen, Grafiken, Formulare und Diagramme. Im Medienmodell werden Klassen für Berichte, Kapitel, Abschnitte, Seiten, Inhaltsverzeichnisse und Fenster bereitgestellt. Objekte dieser Klassen können ihrerseits Objekte des Darstellungsmodells aufnehmen.

Objekte der Teilmodelle können über spezielle Verknüpfungsobjekte (sogenannte Transformatoren und Projektoren) verknüpft werden. Die Verknüpfung zwischen den Teilmodellen erfolgt ausschließlich auf der Ebene der generalisierten Klassen.

Als Entwicklungsplattform wird Unix eingesetzt. Die Implementierung erfolgt trotz des umfassenden Einsatzes objektorientierter Konzepte in der Sprache C. Die Datenhaltung basiert auf einer selbstentwickelten objektorientierten Datenbank.

Seit ihrer Publikation gilt die Arbeit hinsichtlich der rechnergestützten Tragwerksplanung als das Standardwerk. Dennoch ist eine Umsetzung des Konzeptes in ein kommerzielles Produkt bislang nicht erfolgt.

Eine wesentliche Ursache dafür liegt in dem enormen Entwicklungsaufwand, der für ein kommerzielles Produkt zu erbringen wäre. Da auf die Einbeziehung existierender kommerziell verfügbarer Funktionalität von Betriebssystemen und Anwendungen grundsätzlich verzichtet wurde und folglich Funktionalität zur Textverarbeitung, zur Datenhaltung in einer objektorientierten Datenbank sowie die üblicherweise vom Betriebssystem bereitgestellten Dienste für Verbunddokumente selbst implementiert werden mussten, umfasst bereits die Pilotimplementierung mit einem eingeschränkten Funktionsumfang und einer kleinen Auswahl von unterstützten Bauteilen ungefähr 500.000 Programmzeilen. Der Aufwand für ein kommerziellen Anforderungen genügendes System, mit dem die im Ingenieuralltag auftretenden vielfältigen Aufgabenstellungen bearbeitet werden können, dürfte den genannten Umfang um ein Vielfaches übersteigen. Ein derartiger Aufwand kann nicht mehr von einem einzelnen Software-Hersteller erbracht werden und ist auch durch das zu erwartende Marktvolumen in keiner Weise gerechtfertigt. Außerdem würde ein solches System eine reine Insellösung darstellen.

Unter diesem Gesichtspunkt macht sich der monolithische Aufbau des Systems nachteilig bemerkbar. Für jede Erweiterung des Funktionsumfangs muss der Quell-Code zur Verfügung gestellt werden, was mit einer weitgehenden Offenlegung des zugrunde liegenden Know-hows verbunden wäre. Eine Erweiterung durch externe Entwickler ist unter diesen Umständen nicht realistisch. Grundsätzlich stellt sich die Frage, wie extern entwickelte Teile in das Gesamtmodell integriert werden könnten.

Dieselben Schwierigkeiten, die bei einer Erweiterung des Modells durch unabhängige Entwickler auftreten, schränken auch die Möglichkeiten der Anpassung durch den Anwender ein. Zwar ist eine Anpassbarkeit durch die Möglichkeit der Implementierung zusätzlicher spezialisierter Klassen prinzipiell gegeben. Dazu ist jedoch neben der Kenntnis des allgemeinen Konzeptes ein detailliertes Verständnis der Klassenhierarchien erforderlich. Dies kann jedoch in Anbetracht des Programmumfangs von einem Anwender nicht erwartet werden.

Die Nutzeroberfläche des Statischen Editors ist durch eine Vielzahl von Editoren für spezielle Aufgaben gekennzeichnet. Eine effiziente Anwendung des Systems setzt somit voraus, dass dem Anwender die Anwendungsgebiete und die Funktionsweisen der individuellen Editoren bekannt ist. Weiterhin bewirkt die fehlende Berücksichtigung von Standard-Software bekannter Konzepte eine ungewohnte Nutzerführung. Folglich muss mit einem hohen Einarbeitungsaufwand gerechnet werden.

2.3 Werkle et al., 1996

In (Werkle et al., 1996), (Hansen, Röder, 1996), (Werkle et al.,1997) und (Werkle, 1997) ist ein weiterer Ansatz beschrieben, wie sich Berechnung und Bemessung von Tragwerken sowie deren Dokumentation in eine einheitliche Arbeitsumgebung integrieren lassen.

Das entwickelte System besteht aus einem zentralen Projekt-Manager und individuellen Programmen zur Bearbeitung einzelner Bauteile, die über eine objektorientierte Datenbank miteinander gekoppelt sind. Die Implementierung erfolgt objektorientiert in der Programmiersprache C++ auf der Windows-Plattform. Zur Realisierung der Nutzeroberfläche werden die Microsoft Foundation Classes (MFC) genutzt.

Mit dem Projekt-Manager können projektweit gültige Daten wie Materialkennwerte oder Lastannahmen verwaltet, die individuellen Programme aufgerufen und ein Tragwerksbericht generiert werden.

Zur Unterstützung der Entwicklung der individuellen Programme wird ein Class Framework implementiert, das Funktionalität für die Analyse und Bemessung von Tragwerken sowie zur Anbindung der objektorientierten Datenbank bereitstellt. Das Tragwerksmodell ist nicht detailliert dokumentiert, folgt aber mit Klassen für Bauteile und Klassen für deren Kopplungen den gängigen Ansätzen.

Instanzen von Klassen des selbst implementierten Class Frameworks werden in der objektorientierten Datenbank persistent gespeichert. Zur Dokumentation der einzelnen Positionen werden Sichten auf den Datenbestand der Projektdatenbank definiert, die in separaten Dateien gespeichert werden. Prinzipiell könnten die Informationen aus den separaten Dateien ebenfalls in der Projektdatenbank gespeichert werden. Da die MFC-Klassen aber auf eine Datenhaltung in Dateien ausgelegt sind, wären dazu umfangreiche Anpassungen nötig. Außerdem müssten alle potenziell persistenten Klassen in das Datenbankschema eingefügt werden, wodurch zum einen das Datenbankschema stark aufgebläht und unübersichtlich würde. Zum anderen würden Änderungen in einem einzigen Programm das komplette Datenbankschema ungültig machen. Die Daten müssten dann entweder neu erzeugt oder über einen Schemaevolutionsprozess übertragen werden. Eine vollständige Datenhaltung in einer objektorientierten Datenbank ist unter diesen Umständen weder sinnvoll noch praktikabel.

Zur Dokumentation der Tragwerksanalyse werden Verbunddokumente auf der Basis der Technologie Object Linking and Embedding (OLE) genutzt. Die Dokumente des Projekt-Managers wirken dazu als OLE-Container. Wie in Abbildung 2.1 dargestellt, sollen die Sichten auf die Daten der einzelnen Positionen innerhalb einer Projektdatei zu einem einheitlichen Dokument des statischen Berichtes zusammengefügt werden können. Neben den von den individuellen Programmen generierten Textblöcken, die aus Sicherheitsgründen nicht geändert werden können, soll der Nutzer zusätzliche Erläuterungen als freien Text sowie beliebige andere OLE-Objekte einfügen können.

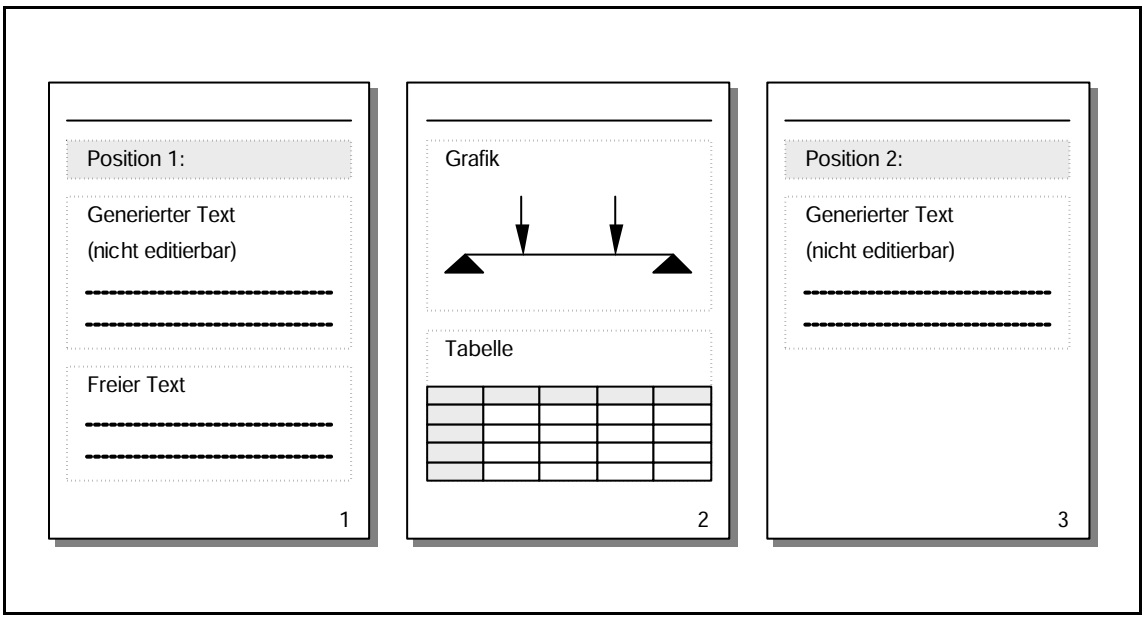


Abbildung 2.1: Struktur des Tragwerksberichtes nach (Werkle et al., 1996)

Ursprünglich war vorgesehen, dass die Positionen der Tragwerksplanung mit Bauteilen eines dreidimensionalen CAD-Gebäudemodells verknüpft werden können. Damit wäre eine Übernahme von Geometriedaten und bei Änderungen eine Erkennung von Inkonsistenzen möglich gewesen. Dieses Merkmal wurde jedoch nicht weiter verfolgt. Als Ursache werden Verzögerungen des Kooperationspartners bei der Entwicklung eines modellorientierten CAD-Systems angeführt, das zur Datenhaltung ebenfalls eine objektorientierte Datenbank einsetzt. Unabhängig davon stellt die automatische Ableitung eines Tragwerkmodells aus einem dreidimensionalen Modell der Gebäudegeometrie ein noch nicht vollständig erforschtes Problem dar. Die Unwägbarkeiten und die Komplexität des Vorhabens wären bei Inangriffnahme dieses Funktionsmerkmals nochmals deutlich gesteigert worden.

Weiterhin wird darauf hingewiesen, dass der Lastabtrag zukünftig den Regeln des Eurocodes EC 1 genügen muss. Damit wird die Komplexität des Lastabtrages nochmals wesentlich gesteigert. Um die Teilsicherheits- und Kombinationsbeiwerte korrekt berücksichtigen zu können, müssen die Lastanteile strikt nach ständigen Lasten, Verkehrslasten und außergewöhnlichen Lasten getrennt werden.

Zusammenfassend muss festgestellt werden, dass die ursprünglich angestrebten Ziele mit dem beschrittenen Weg nicht im vollen Umfang realisiert werden konnten.

2.4 Hinz, 1998

In (Hinz, 1998) wird ein Konzept für ein allgemein gültiges Tragwerksmodell vorgestellt und unter Einsatz von komponentenorientierter Software-Technik umgesetzt. Zunächst wird herausgearbeitet, dass die bisherigen Konzepte vom Ziel geprägt waren, über eine objektorientierte Modellierung einen automatischen Lastabtrag zu realisieren.

Ein auf einem objektorientierten Modell basierender automatischer Lastabtrag setzt jedoch voraus, dass alle relevanten Bauteile einschließlich aller Verbindungen klassifiziert wurden. Tragstrukturen, die Bauteile oder Verbindungen enthalten, die nicht im Modell enthalten sind, können damit nicht erfasst werden. Im Anschluss daran wird anhand eines Beispiels gezeigt, dass Tragwerksmodelle zwar verhältnismäßig einfach auf der Ebene der Klassen formuliert werden können, sich diese jedoch nur mit Einschränkungen oder nicht eindeutig auf die Ebene der Objekte übertragen lassen. Zusammenfassend wird festgestellt, dass Tragwerksmodelle, die auf einer Klassifizierung von Bauteilen und deren Kopplungen beruhen, keine allgemein gültige Lösung darstellen können.

Aus diesen Überlegungen heraus wird ein neues Konzept für ein Tragwerksmodell vorgeschlagen, bei dem der Lastabtrag nicht mehr durch ein vorgegebenes Regelwerk automatisch generiert werden soll. Stattdessen legt der Ingenieur den Lastabtrag bei der Bearbeitung nach seinen Vorstellungen fest, indem er relevante Ergebniswerte einer Position mit den zugehörigen Parametern von anderen Positionen verknüpft. Die erstellten Verknüpfungen werden vom System protokolliert und führen somit zu einem zur Bearbeitungszeit definierten Tragwerksmodell. Das Konzept wird deshalb als Protokollierung von Verknüpfungen bezeichnet.

Vor der Durchführung der Analyse wird das Tragwerk auf herkömmliche Art und Weise in Positionen zerlegt. Diese Positionen werden unabhängig voneinander durch individuelle Programme analysiert und dokumentiert. Ein derartiger Vorgang wird als Prozesskette bezeichnet und findet auf der sogenannten Berechnungsebene statt. Um die einzelnen Prozessketten zu verknüpfen werden Abbilder der Ergebnis- und Eingabewerte auf eine sogenannte Verknüpfungsebene projiziert. Auf dieser Ebene können die Verknüpfungen unabhängig von den eingesetzten Berechnungsmodellen und -verfahren allgemein gültig protokolliert werden.

Der auf der Verknüpfungsebene definierte Verknüpfungsgraph stellt einen gerichteten azyklischen Graphen dar. Nach Änderungen können mit Hilfe dieses Graphen die davon betroffenen Prozessketten ermittelt und auf Wunsch des Bearbeiters aktualisiert werden. Durch eine vorgeschaltete topologische Sortierung wird sichergestellt, dass ein einziger Durchlauf zur Aktualisierung genügt. Außerdem wird durch einen Ablauf des Algorithmus der Tiefensuche verhindert, dass Zyklen in den Verknüpfungsgraph eingefügt werden können.

Zur eindeutigen Identifizierung der Informationen wird ein zweistufiger Schlüssel verwendet. Im ersten Schritt wird über den Strukturschlüssel genau eine Teilstruktur identifiziert. Im zweiten Schritt wird über den Informationsschlüssel die zu übertragende Information identifiziert. Zu diesem Zweck verwaltet jedes Teilstrukturobjekt eine Liste, in der die von ihm erzeugten Informationen enthalten sind.

Ein weiterer Schwerpunkt der Arbeit liegt in der Formulierung von Strategien zur effizienten Umsetzung des Konzeptes der Protokollierung von Verknüpfungen mit den aktuellen Methoden der Software-Technik. Es wird erkannt, dass eine Trennung von langlebigen (numerische Berechnungen) und kurzlebigen (Nutzeroberfläche) Software-Bausteinen mit großen Vorteilen verbunden ist.

Aus diesem Grund wird die Funktionalität zur Berechnung von Stabtragwerken in eine separate Software-Komponente ausgegliedert. Diese Software-Komponente wird als Berechnungsdienst bezeichnet und erlaubt es, dieselbe Funktionalität in mehrere Anwendungen mit unterschiedlichen Nutzeroberflächen einzubinden.

Es wird weiterhin demonstriert, dass sich auch sogenannte Altsysteme über das Konzept der Komponentenschalen mit neuen Nutzeroberflächen versehen lassen. Aufgabe der Komponentenschale ist es dabei, über ein Interface die Erzeugung der notwendigen Eingabedateien zu ermöglichen. Im Anschluss daran wird das Altsystem gestartet und die Ausgabedateien erzeugt. Diese werden durch die Komponentenschale interpretiert, wobei die Ergebnisse über das Interface abgefragt werden können.

Die Implementierung findet auf der Windows-Plattform statt und erfolgt objektorientiert in der Sprache C++ unter Nutzung der Microsoft Foundation Classes. Während die Datenhaltung der einzelnen Prozessketten in proprietären Dateien erfolgt, wird für die Verwaltung der Verknüpfungen die objektorientierte Datenbank Poet verwendet.

Die Arbeit stellt einen großen Fortschritt dar, da erstmalig Funktionalität in wiederverwendbare, binäre Software-Komponenten ausgegliedert wird. Da diese Software-Komponenten Black-Box-Bausteine darstellen, können sie externen Entwicklern zugänglich gemacht werden, ohne dass damit eine Offenlegung von Know-how verbunden ist.

Eine genauere Untersuchung der Arbeit offenbart jedoch, dass lediglich die Funktionalität zur numerischen Berechnung von Stabtragwerken und zur Verknüpfung der einzelnen Prozessketten in je eine separate Software-Komponente ausgegliedert wurde. Darüber hinaus werden noch Komponenten zur Dokumentation, zur Visualisierung des Verknüpfungsgraphen und zur Projektverwaltung angesprochen, aber nicht weiter dokumentiert.

Der Umfang des Programms BigSolve zur Analyse von Stabtragwerken wird mit ungefähr 45.000 Codezeilen angegeben. Davon entfallen 4.500 Zeilen auf die Implementierung des Allgemeinen Weggrößenverfahrens als Berechnungsdienst. Die Nutzeroberfläche zur grafisch-interaktiven Bearbeitung ist konventionell objektorientiert programmiert und umfasst 150 Klassen beziehungsweise 40.000 Codezeilen. Der Anteil der Software-Komponenten liegt somit bei ungefähr 10%. Das bedeutet, dass der Einsatz von Software-Komponenten für 90% der Entwicklungsaufgaben keine Vorteile bringt.

Der Autor versucht diesem Mangel durch umfangreichen Einsatz herkömmlicher objektorientierter Strategien zu begegnen. Neben der Nutzung der kommerziell verfügbaren Microsoft Foundation Classes wird ein eigenes Class Framework entwickelt, das spezielle Funktionalität für Tragwerksplanungs-Software bereitstellt. Zusätzlich wird ein Codegenerator entwickelt, der aus den Klassen der MFC und des selbst entwickelten fachspezifischen Class Frameworks ein Programmgerüst erstellt. Dieses Gerüst muss dann vom Entwickler mit problemspezifischem Code ergänzt werden. Mit der Kompilierung werden die Class Frameworks, der durch den Codegenerator generierte und der selbst geschriebene Code zu einer herkömmlichen monolithischen Anwendung verschmolzen.

Eine Ursache für den überraschend hohen Anteil an objektorientiert entwickeltem Code ist wiederum die fehlende Einbindung von Standard-Software. Beispielsweise werden zahlreiche Klassen implementiert, um die benötigten Grundfunktionalitäten einer Textverarbeitung bereitzustellen.

Eine Anpassbarkeit der eigentlichen Anwendungen ist infolge des monolithischen Aufbaus nicht gegeben. Allerdings kann die Funktionalität der Anwendungen zumindest teilweise über Programmierschnittstellen verfügbar gemacht werden.

Die Nutzeroberfläche ist gekennzeichnet durch individuelle Programme, die jeweils über ein eigenes Dateiformat verfügen. Wie auch bei (Molkenthin, 1994a) lässt dieser Ansatz einen hohen Einarbeitungsaufwand erwarten.

2.5 Schlussfolgerungen

Eine Analyse des Standes der Technik zeigt, dass die vorgestellten Konzepte bislang nur in Ansätzen oder gar nicht bei der Entwicklung von kommerziellen Produkte aufgegriffen wurden. Als wesentliche Ursache dafür wurde der unverhältnismäßig hohe Entwicklungsaufwand identifiziert, der aus den unzureichenden Möglichkeiten zur Wiederverwendung bereits implementierter Funktionalität resultiert. Hinzu kommt die fehlende Erweiterbarkeit der bisherigen Ansätze durch unabhängige Software-Entwickler. Dies bewirkt, dass ein Rückgriff auf die zahlreichen Vorarbeiten nicht möglich ist.

Aus der Sicht des Software-Entwicklers müssen somit an zukünftige Systeme folgende Anforderungen gestellt werden:

- der Anteil an wiederverwendeter Funktionalität muss gesteigert werden
- die Systeme müssen durch externe Entwickler erweiterbar sein

Die genannten Anforderungen dienen lediglich dazu, dass die Entwicklung komplexer Software-Systeme unter software-technischen Gesichtspunkten beherrschbar bleibt. Aus der Sicht des anwendenden Ingenieurs stellt ihre Erfüllung keinen wesentlichen Vorteil dar. Um jedoch auch die Akzeptanz eines neu entwickelten Systems bei den Nutzern sicherzustellen, muss es aus deren Sicht gegenüber den bisherigen Ansätzen Vorteile aufweisen.

Da eine ergonomisch gestaltete Nutzeroberfläche zu einer erhöhten Arbeitsproduktivität bei gleichzeitiger Qualitätssteigerung führen kann, stellt diese für den Anwender ein wesentliches Qualitätsmerkmal dar. Eine ergonomisch gestaltete Nutzeroberfläche ist deshalb ebenso wichtig wie die Robustheit und Effizienz eines Software-Produktes. Unzureichend gelöst ist bislang die Integration von unabhängig voneinander entwickelten Produkten in eine einheitliche Arbeitsumgebung. Außerdem wäre eine weitgehende Anpassbarkeit der Software an die individuellen Bedürfnisse der Anwender wünschenswert. Grundsätzlich würde es einen großen Fortschritt darstellen, wenn die Tragwerksplaner ihre fachlichen Lösungen wieder selbst ohne den gegenwärtig notwendigen Umweg über professionelle Software-Entwickler implementieren und erweitern könnten. Somit werden aus der Perspektive des Anwenders an zukünftige Systeme die folgenden Anforderungen gestellt:

- die Nutzeroberfläche muss ergonomisch gestaltet sein
- der Informationsfluss zwischen den beteiligten Anwendungen muss optimiert werden
- die Arbeitsumgebung muss durch Anwender angepasst werden können
- die software-technischen und fachlichen Aufgaben müssen getrennt voneinander bearbeitet werden können

Kapitel 3
Grundlagen komponentenorientierter Software-Technik

Bereits 1968 prophezeite Doug McIlroy, dass die sogenannte "Software-Krise" durch den Einsatz massenproduzierter Software-Komponenten beendet werden würde (McIlroy, 1968). In der Tat stellt die komponentenorientierte Software-Technik den gegenwärtig aussichtsreichsten Ansatz zur Erfüllung zahlreicher aktueller Anforderungen der Software-Industrie dar. Ziel dieses Kapitels ist es, durch Definition der verwendeten Begriffe und Erläuterung der relevanten Konzepte eine gemeinsame Sprache als Grundlage für das Verständnis der nachfolgenden Ausführungen zu schaffen. Zwangsläufig können viele Themen nicht erschöpfend behandelt werden; für eine umfassende Diskussion sei auf (Szyperski, 1998) als aktuelles Standardwerk verwiesen.

3.1 Definitionen

Für eine Diskussion komponentenorientierter Software-Technik ist die Definition der Begriffe Software-Komponente und Interface von herausragender Bedeutung. Während Software-Komponenten die eigentlichen Bauelemente bilden, stellen Interfaces deren mögliche Verbindungsstellen dar.

3.1.1 Software-Komponente

Für den Begriff Software-Komponente existieren gegenwärtig eine Vielzahl sich teilweise widersprechender Definitionen, die an dieser Stelle nicht alle berücksichtigt werden können. Im Rahmen dieser Arbeit wird der Begriff Software-Komponente entsprechend der 1996 auf der European Conference on Object-Oriented Programming (Szyperski, Pfister, 1997) formulierten Definition verwendet:

> *"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."*

Die wesentlichen Aussagen dieser Definition sind, dass Software-Komponenten unabhängig voneinander entwickelt und vertrieben sowie schließlich durch Dritte zu funktionsfähigen Anwendungen zusammengesetzt werden. Software-Komponenten werden dabei als vorgefertigte Bauelemente betrachtet, die vor Ort an die speziellen Bedürfnisse der Anwendung angepasst werden können. Impliziter Bestandteil der obigen Definition ist, dass Software-Komponenten in binärer Form vorliegen müssen. Das Zusammenfügen der Software-Komponenten wird durch für Anbieter und Nutzer verbindliche Interfaces erlaubt. Die gegenseitigen Abhängigkeiten der Software-Komponenten müssen sich auf den ausdrücklich spezifizierten Kontext beschränken. Die Kontextabhängigkeiten beinhalten, welche Interfaces für eine Zusammenarbeit von Software-Komponenten erforderlich sind und welche Plattformen unterstützt werden.

3.1.2 Interface

Interfaces bilden die Verbindungsstellen, über die unabhängige Software-Komponenten zu einem System verbunden werden können. Technisch gesehen stellt ein Interface eine Menge von benannten Diensten dar, die von den Nutzern einer Software-Komponente angefordert werden können. Die Semantik aller Dienste muss derart spezifiziert werden, dass eindeutig festgelegt ist, welche Funktionalität vom Anbieter der Software-Komponente zu implementieren ist und wie ein Nutzer dieser Software-Komponente auf diese Dienste zugreifen kann.

Die Spezifikation eines Interfaces stellt somit einen Vertrag zwischen Anbieter und Nutzer dar. Auf der Grundlage dieses Vertrags können mehrere Anbieter Software-Komponenten entwickeln, welche die Vereinbarungen des Vertrags erfüllen. Die Nutzer von Software-Komponenten können ihrerseits von deren vertragsgemäßem Verhalten ausgehen. Ein Interface und dessen Spezifikation dient folglich als Mittler zwischen mehreren Anbietern und Nutzern von Software-Komponenten (Abbildung 3.1).

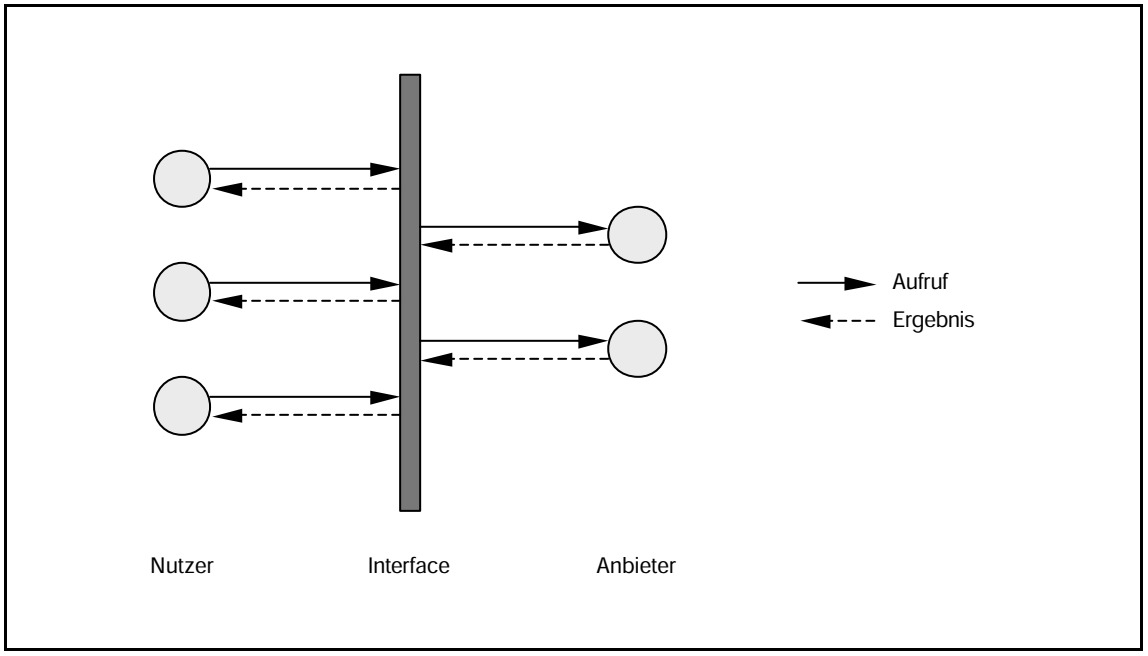


Abbildung 3.1: Ein Interface als Mittler zwischen mehreren Anbieter und Nutzern von Software-Komponenten

3.2 Entwicklung der Programmierparadigmen

Die Software-Industrie erfährt seit Jahrzehnten ein exponentielles Wachstum. Den sich daraus ergebenden Herausforderungen versucht sich die Software-Industrie durch adäquate technologische Verbesserungen zu stellen. Im Laufe der Zeit erfolgte deshalb eine kontinuierliche Evolution der eingesetzten Softwaretechnologien: beginnend mit Maschinensprachen und später Assembler- und Hochsprachen, über strukturierte und modulare zur objektorientierten Programmierung und schließlich zur komponentenorientierten Programmierung.

Mit der Entwicklung der objektorientierten Softwaretechnik schien man viele der offensichtlichen Probleme der Software-Industrie adressiert zu haben. Als besonders vielversprechend wurden die Möglichkeiten der Wiederverwendung von Code eingeschätzt, die man glaubte, durch extensive Anwendung von Vererbung erreichen zu können. Als äußerst nützlich wurde dabei empfunden, dass eine Verbesserung einer Klasse automatisch mit einer Verbesserung der abgeleiteten Klassen einhergeht. Nach einigen Jahren praktischer Anwendung werden jedoch auch die damit verbundenen Probleme erkannt (vergleiche Kapitel 3.3). Aktuelle Erkenntnisse deuten darauf hin, dass eine effiziente Wiederverwendung von Software-Bausteinen am ehesten durch möglichst unabhängige, in binärer Form vorliegende Software-Komponenten erreicht werden kann.

3.2.1 Objektorientierte Programmierung

Bei der objektorientierten Programmierung wird ein Sachverhalt durch eine Menge von miteinander kommunizierenden Objekten abgebildet. Die einzelnen Objekte sind Instanzen von Klassen, wobei Klassen Typbeschreibungen von gleichartigen Objekten darstellen. Die Objekte müssen entsprechend der klassischen Definition die drei wesentlichen Konzepte der objektorientierten Programmierung unterstützen:

- Kapselung
- Polymorphie
- Vererbung

Die Forderung nach einer Kapselung resultiert primär aus dem Wunsch nach einer Erhöhung des Abstraktionsniveaus. Unter Kapselung wird dabei verstanden, dass sämtliche Details über den internen Zustand und die Implementierung der Funktionalität eines Objektes nach außen hin verborgen sind. Der Zugriff auf die den Zustand eines Objektes beschreibenden Eigenschaften ist ausschließlich über seine öffentlichen Methoden möglich. Die Menge der dem Zugriff auf die Eigenschaften eines Objektes dienenden Methoden kann dabei als Interface betrachtet werden. Die Instanzen einer Klasse können über dieses Interface ohne Kenntnis der internen Konzepte als abstrakte Objekte angesprochen werden. Darüber hinaus werden unbeabsichtigte Manipulationen des Objektzustands verhindert.

Polymorphie bedeutet, dass Objekte, die Instanzen unterschiedlicher Klassen sind, gleich behandelt werden können. Das heißt, Methoden können ohne Kenntnis des konkreten Objekttyps gleichartig angesprochen werden. Das Verhalten der Objekte wird dabei durch die jeweilige Implementierung der Methoden bestimmt. Das polymorphe Verhalten der Objekte kann beispielsweise durch Vererbung realisiert werden.

Vererbung stellt das dominierende Mittel der objektorientierten Programmierung zur Wiederverwendung von Code dar. Zu diesem Zweck können Klassen von anderen Klassen abgeleitet werden. Die abgeleiteten Klassen werden als Subklassen bezeichnet und erben die Eigenschaften und Methoden der Superklassen genannten vererbenden Klassen. Folglich sind die Instanzen der Subklassen polymorph zu den Instanzen der Superklasse. Die Wiederverwendung von Code wird realisiert, indem Funktionalität, die in mehreren Klassen benötigt wird, in einer Superklasse zentral implementiert wird.

Zu den drei erwähnten Konzepten tritt häufig noch das Konzept des späten Bindens. Die Notwendigkeit des späten Bindens ergibt sich aus dem Konzept der Polymorphie. Da Objekte verschiedener Klassen unter Umständen polymorph behandelt werden können, kann die erforderliche Implementierung der Methoden erst zur Laufzeit ermittelt und gebunden werden.

Aufgrund der diskutierten innovativen Konzepte stellt die objektorientierte Programmierung eine sehr leistungsfähige Technologie dar. Trotz der mittlerweile erreichten Akzeptanz und Verbreitung weist die objektorientierte Software-Technologie die zuvor beschriebenen Schwächen auf, die in einem weitern Entwicklungsschritt durch die komponentenorientierte Software-Technologie behoben werden sollen.

3.2.2 Komponentenorientierte Programmierung

Bei der komponentenorientierten Programmierung wird ein Problem durch eine Menge von miteinander kommunizierenden Software-Komponenten abgebildet. Die einzelnen Software-Komponenten sind im Kontext dieser Arbeit in binärer Form vorliegende Einheiten. Eine Definition von komponentenorientierter Programmierung, die sich an der Art gebräuchlicher Definitionen objektorientierter Programmierung anlehnt, ist in (Szyperski, 1995) und (Szyperski, 1998) angegeben. Danach müssen Software-Komponenten die folgenden vier wesentlichen Konzepte unterstützen:

- Polymorphie
- Modulare Kapselung
- spätes Laden und Binden
- Sicherheit

Dabei wird die Forderung nach Polymorphie als besonders entscheidend betrachtet. Polymorphie bedeutet in diesem Zusammenhang, dass Software-Komponenten beliebig durch andere Software-Komponenten ersetzt werden können, solange diese über dasselbe Interface und eine der Spezifikation des Interfaces entsprechende Implementierung verfügen.

Unter modularer Kapselung wird eine Kapselung auf der Ebene der Software-Komponente verstanden, gegebenenfalls also über eine Vielzahl von Objekten. Zu diesem Zweck werden Interfaces und Implementierung einer Software-Komponente voneinander getrennt. Alle Interaktionen von Software-Komponenten nach außen erfolgen ausschließlich über publizierte Interfaces. Die Implementierung ist somit vollständig gekapselt, da Software-Komponenten ausschließlich in binärer Form vorliegen. Als Resultat wird ein im Vergleich zur objektorientierten Programmierung wesentlich höherer Grad der Kapselung von Informationen erreicht.

Nach ihrer Definition müssen Software-Komponenten unabhängig voneinander vertrieben und vor Ort zu einem lauffähigen System zusammengefügt werden können. Es muss möglich sein, dass eine Software-Komponente erst zur Laufzeit lokalisiert, geladen und gebunden wird. Ein derartiger Mechanismus wird als spätes Laden und Binden bezeichnet.

Bei der traditionellen Software-Entwicklung werden Module einzeln entwickelt und getestet. Aus diesen Modulen wird das Programm erstellt und das Zusammenspiel der Module getestet. Somit können auch Fehler erkannt werden, die aus dem fehlerhaften Zusammenspiel der Module resultieren.

Da Software-Komponenten aufgrund unabhängiger Entwicklung und unabhängigen Vertriebs erst durch spätes Binden zusammengefügt werden, können Probleme, die nur am konkreten Gesamtsystem auftreten, nicht vorab durch die Hersteller erkannt und beseitigt werden. Eine Fehlfunktion einer Software-Komponente kann die Funktionsfähigkeit der gesamten Anwendung beeinträchtigen. Folglich ist ein komponentenbasiertes System nur so robust wie seine schwächste Software-Komponente. Darüber hinaus können Fehlerursachen in einem System unabhängiger Software-Komponenten häufig nicht eindeutig identifiziert und zugeordnet werden. Da die Konsequenzen eines Versagens einer Software-Komponente nicht abgeschätzt werden können, muss der Gesichtspunkt der Sicherheit wesentlich stärker als bisher berücksichtigt werden. Diesem erhöhten Sicherheitsbedürfnis wird durch die Forderung nach Typ- und Modulsicherheit Rechnung getragen (Szyperski, 1998).

Unter Typsicherheit werden dabei die dynamische Überprüfung potenziell gefährlicher Operationen, die nicht zum Zeitpunkt der Kompilierung erkannt werden können (z.B. ungültige Indizes in Feldern) oder ein als Garbage Collection bezeichnetes automatisches Speichermanagement verstanden.

Modulsicherheit bedeutet, dass es einer Software-Komponente nicht möglich sein sollte, ohne Berechtigung auf andere Software-Komponenten oder Dienste des Betriebssystems zuzugreifen. Dies setzt voraus, dass ein Mechanismus existiert, der überprüft, ob die Nutzung der angeforderten Dienste zulässig ist.

Beim Vergleich der Paradigmen objekt- und komponentenorientierter Programmierung fällt auf, dass das Konzept der Vererbung durch die Konzepte des Späten Bindens sowie der Typ- und Modulsicherheit ersetzt wurde. Dies erscheint zunächst dem Anspruch der komponentenorientierten Programmierung zu widersprechen, eine effiziente Wiederverwendung von Programm-Code zu ermöglichen. Das Konzept der Vererbung stellt lediglich ein Mittel zur Wiederverwendung von Programm-Code dar und kann folglich durch andere Mechanismen wie die Komposition von Software-Komponenten ersetzt werden (Brockschmidt, 1995).

3.3 Gegenüberstellung relevanter Begriffe und Konzepte

Sowohl die objektorientierte als auch die komponentenorientierte Software-Technik erheben im Kern den Anspruch, die Entwicklung von wiederverwendbaren Software-Bausteinen zu ermöglichen. Die gleiche Zielstellung bewirkt, dass häufig nur die Parallelen zwischen beiden Konzepten zur Kenntnis genommen werden. Die gleichzeitig vorhandenen fundamentalen Unterschiede bezüglich der Kapselung der Informationen und der Mechanismen zur Wiederverwendung von Programm-Code werden dabei häufig verwischt. Im Folgenden sollen deshalb die Unterschiede bezüglich der relevanten Begriffe und Konzepte herausgestellt werden.

3.3.1 White-Box-Abstraktion versus Black-Box-Abstraktion

Die Begriffe White-Box-Abstraktion und Black-Box-Abstraktion beziehen sich auf die Sichtbarkeit der Implementierung.

Bei der objektorientierten Programmierung bewirkt die Kapselung durch das Interface, dass der interne Zustand eines Objektes von außen nur über öffentliche Methoden modifiziert werden kann. Die Implementierung selbst ist nicht gekapselt und somit vollständig sichtbar und kann vom Entwickler zur Erweiterung seines Verständnisses der Abstraktion studiert werden. Eine derartige Abstraktion wird als White-Box-Abstraktion bezeichnet (Abbildung 3.2).

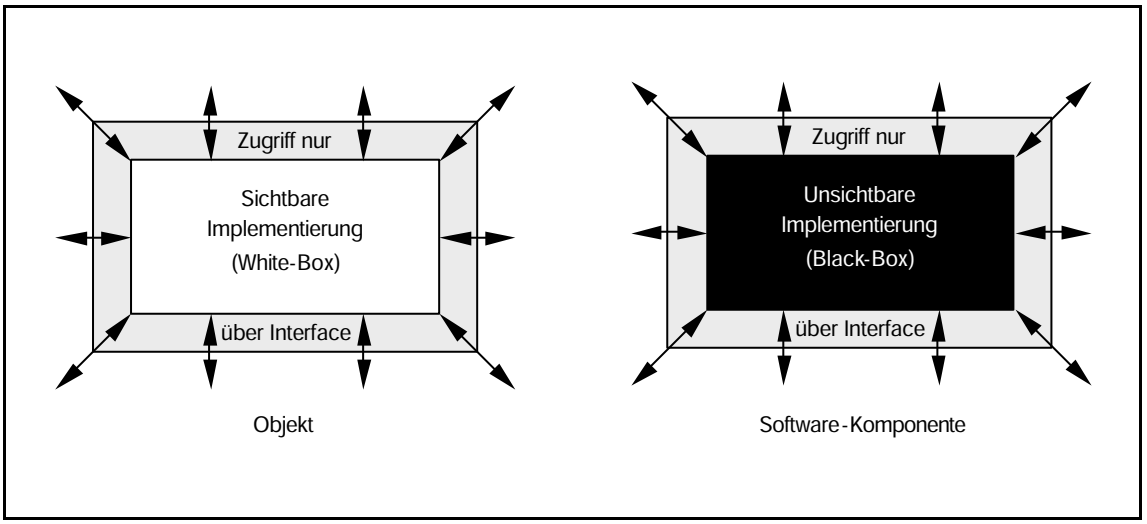


Abbildung 3.2: Sichtbarkeit der Implementierung bei Objekten und Software-Komponenten

Die Sichtbarkeit der Implementierung führt dazu, dass eine gravierende Schwäche der objektorientierten Programmierung verdeckt wird. Der Einsatz von Klassenbibliotheken erfordert vom Programmierer ein umfassendes Verständnis der durch die Klassenhierarchie vorgenommenen Abstraktionen. Diese Ansicht wird durch die Praxis der Hersteller von Klassenbibliotheken bestätigt, dem Kunden den gesamten Quell-Code zur Verfügung zu stellen. Wenn jedoch die Implementierung die einzig vollständige Dokumentation einer Klassenbibliothek darstellt, dann wurde der Anspruch der Entkopplung von Anbieter und Nutzer eines Software-Bausteins durch ein Interface nicht erreicht.

Durch Studium des Quell-Codes können sich Entwickler ein detailliertes Verständnis der Implementierung erwerben, das über die vom Interface und dessen Spezifikation bereitgestellten Informationen weit hinausgeht. Die Ausnutzung dieses Wissens führt zu einer höheren Abhängigkeit von der Implementierung. Bei einer Änderung der Implementierung besteht die Gefahr, dass die Objekte untereinander nicht mehr kompatibel sind. In (Szyperski, 1995) wird deshalb geschlussfolgert, dass extensiver Einsatz von Vererbung und Kapselung von Informationen zwei sich gegenseitig ausschließende Konzepte sind.

Es ist offensichtlich, dass das Ziel der Kapselung auf diese Art nicht im vollen Umfang erreicht werden kann. Aus diesem Grund werden Black-Box-Abstraktionen propagiert. Bei einer idealen Black-Box-Abstraktion sind keinerlei Implementierungsdetails nach außen sichtbar (Abbildung 3.2). Black-Box-Abstraktionen können beispielsweise durch binäre Software-Komponenten realisiert werden. Entwickler können sich ausschließlich auf Interfaces und deren Spezifikation verlassen. Eine Spezifikation für Black-Box-Komponenten muss zwangsläufig wesentlich präziser ausfallen, da der Quell-Code nicht mehr als Ergänzung der Spezifikation dienen kann. Als Konsequenz der Black-Box-Abstraktion kann erwartet werden, dass sich die wechselseitigen Abhängigkeiten der Software-Komponenten wesentlich verringern.

Darüber hinaus stellt die Black-Box-Abstraktion einen wesentlichen Schritt hin zur Etablierung eines Marktes für Software-Komponenten dar. Bislang wurden die Hersteller von herkömmlichen objektorientierten Klassenbibliotheken gezwungen, neben dem eigentlichen Produkt auch in Form des Quell-Codes Teile des zur Herstellung dieses Produktes erforderlichen Know-hows zu verkaufen.

Mit Einschränkungen war es allerdings auch bisher mit den herkömmlichen Mitteln der objektorientierten Software-Technik möglich, eine Black-Box-Abstraktion zur verwenden. Zu diesem Zweck wurde der wiederzuverwendende Teil der Implementierung in eine als Dynamic Link Library (DLL) bezeichnete binäre Bibliothek kompiliert. Mit diesem Behelf waren jedoch zahlreiche Schwierigkeiten in Kauf zu nehmen: So führt nahezu jede Modifikation der Implementierung zu einem veränderten Speicher-Layout der DLL. Folglich muss in den meisten Fällen nach dem Erstellen einer neuen Version auch die Client-Anwendung neu kompiliert werden. Wenn dies aufgrund der großen Anzahl von Clients oder einer Nutzung durch Dritte nicht ohne weiteres möglich ist, muss eine neue Version der Bibliothek ausgeliefert werden. Da die älteren Versionen jedoch für ältere Client-Anwendungen weiterhin benötigt werden, muss die Funktionalität, die eigentlich durch Wiederverwendung zu einem kompakteren Programm-Code führen sollte, sogar mehrfach vorhanden sein.

Neben der Versionsproblematik bleiben weitere Einschränkungen wie Sprach- und Compiler-Abhängigkeit bestehen. Die Wiederverwendung von objektorientiert geschriebener Software ist somit lediglich im Entwickler-Team oder innerhalb einer Organisation realistisch. Unter diesen Randbedingungen war das mögliche Volumen eines Marktes für Software-Komponenten zwangsläufig gering.

3.3.2 Evolution versus Unveränderlichkeit von Interfaces

Ein Interface stellt – wie schon erwähnt – gemeinsam mit seiner Spezifikation eine Art Vertrag dar, auf dessen Grundlage durch Software-Komponenten Dienste zur Verfügung gestellt oder genutzt werden können. Nachdem ein Vertrag veröffentlicht wurde, kann dieser nur durch einen von allen Beteiligten getragenen Beschluss verändert werden. Die daraus resultierende faktische Unveränderlichkeit der Geschäftsbeziehungen ist für die Software-Industrie als eine von Innovationen und Veränderungen geprägte Branche nicht akzeptabel. Verträge müssen folglich in irgendeiner Form modifizierbar sein.

Dabei lassen sich zwei Arten der Änderung eines Vertrages zwischen Software-Komponenten unterscheiden. Ändert sich die Spezifikation des Interfaces, so stellt dies eine semantische Änderung dar. Ändert sich das Interface selbst, so spricht man von einer syntaktischen Änderung. Jede Änderung des Vertrages hat Konsequenzen für die durch den Vertrag gebundenen Parteien. Die Konsequenzen von Vertragsänderungen waren prinzipiell beherrschbar, solange die Software innerhalb eines Teams als monolithische Anwendung nach den Paradigmen der objektorientierten Programmierung entwickelt wurde. Mit zunehmender Projektgröße wurde jedoch die Wiederverwendung von Codefragmenten immer notwendiger. Zu diesem Zweck wurden durch externe Anbieter Klassenbibliotheken wie die Microsoft Foundation Classes entwickelt, in denen häufig benötigte Funktionalität gekapselt ist. Entwickler können diese Funktionalität nutzen und an ihre konkreten Bedürfnisse anpassen, indem sie neue Klassen von den existierenden Klassen der Klassenbibliothek ableiten.

In einer daraus resultierenden Vererbungshierarchie weisen die Subklassen zwangsläufig starke Abhängigkeiten von den Superklassen auf. Da die Klassenbibliothek und die konkrete Implementierung unabhängig voneinander entwickelt werden, ergibt sich mit dem Erscheinen einer neuen Version der verwendeten Klassenbibliothek die Frage, ob Superklassen und Subklassen noch miteinander verträglich sind. Die mit der unabhängigen Evolution von Super- und Subklassen verbundenen Schwierigkeiten werden in der Literatur als Fragile-Base-Class-Problem bezeichnet (Brockschmidt, 1995).

Jede Änderung der Spezifikation der Superklassen kann umfangreiche Änderungen der davon betroffenen Subklassen erforderlich machen. Die aus den veränderten Spezifikationen resultierenden Schwierigkeiten werden unter dem Begriff semantisches Fragile-Base-Class-Problem zusammengefasst. Eine Lösung des semantischen Fragile-Base-Class-Problems ist zur Zeit nicht absehbar. Daneben existiert auch ein syntaktisches Fragile-Base-Class-Problem, welches ausschließlich Konsequenzen der Modifikation von Interfaces adressiert. Rein syntaktische Modifikationen, wie das Einfügen von zusätzlichen Vererbungsebenen oder ein Verschieben von Methoden entlang der Vererbungshierarchie, sollen kein neues Kompilieren erfordern, solange die Parameter und Rückgabewerte unverändert bleiben. Vereinzelt wird das syntaktische Fragile-Base-Class-Problem als gelöst deklariert (IBM, 1994). Allerdings bleiben zahlreiche Änderungen syntaktischer Natur, wie die Veränderung einer Parameterliste oder das Zusammenfassen von zwei Methoden zu einer Methode und umgekehrt, dabei unberücksichtigt.

Änderungen von Interfaces und deren Spezifikationen stellen in einer objektorientierten Umgebung ein gravierendes, aber mit großem Aufwand lösbares Problem dar. Sobald jedoch Software-Komponenten auf dem Markt gehandelt werden, geht jegliche Kontrolle über die zugrunde liegenden Verträge verloren. Nachträgliche Vertragsänderungen sind deshalb von vornherein ausgeschlossen. Unter diesen Randbedingungen gibt es nur eine Konsequenz für die komponentenorientierte Softwareentwicklung: Interfaces und deren Spezifikationen müssen ab dem Zeitpunkt ihrer Publikation als unveränderlich betrachtet werden. Jede Modifikation des Vertrages erfordert die Definition eines neuen Interfaces. Damit eine Software-Komponente dennoch verschiedene Versionen unterstützen kann, muss sie eine beliebige Anzahl von Interfaces implementieren können.

3.3.3 Vererbung versus Komposition

Vererbung wird in der objektorientierten Programmierung angewendet, um einerseits die Wiederverwendung von Codefragmenten zu ermöglichen und andererseits Objekte polymorph behandeln zu können. Während die Wiederverwendung von Code durch eine Vererbung der Implementierung erreicht wird, ermöglicht die Vererbung des Interfaces ein polymorphes Verhalten. Somit ergeben sich mit Implementierungs- und Interface-Vererbung zwei Varianten der Vererbung.

Der Einsatz des Konzeptes der Implementierungsvererbung ist wegen der dann nicht mehr möglichen Black-Box-Abstraktion und der daraus resultierenden diffizilen Evolution von Klassenhierarchien für Komponenten-Software offensichtlich nicht geeignet. Zugleich wird durch die verteilte Entwicklung von Software-Komponenten der Nutzen von Interface-Hierarchien fraglich. Wesentlich wichtiger, als durch Ableitung von Supertypen neue Subtypen definieren zu können, erscheint die Möglichkeit, auf durch Dritte definierte Interfaces zurückgreifen zu können. Damit stellt sich die Frage, ob die primären Ziele der Vererbung nicht durch andere, zweckmäßigere und weniger kritische Mechanismen erreicht werden können.

In (Gamma et al., 1995) wird argumentiert, dass sich durch Komposition von Objekten auf einfache Weise mit Implementierungsvererbung vergleichbare Effekte erreichen lassen. Da die Kommunikation zwischen den Objekten ausschließlich über Interfaces erfolgt, wird die Forderung nach Kapselung erfüllt. Darüber hinaus können Objekte beliebig ausgetauscht werden, sofern sie über ein der selben Spezifikation entsprechendes Interface verfügen. Da die Abhängigkeiten der Objekte sich ausschließlich auf Interfaces von anderen Objekten beziehen, ergeben sich wesentlich weniger Abhängigkeiten als beim Einsatz von Vererbung. Werden außerdem Implementierung und Interface voneinander getrennt und Mittel zur Verfügung gestellt, die es einer Klasse erlauben, eine beliebe Zahl von Interfaces zu implementieren, kann dadurch auch die Interface-Vererbung ersetzt werden. Ein derartiger Ansatz wird beispielsweise durch die Sprache Java verfolgt (Sun, 2000).

Obwohl die zuvor genannten Ideen ursprünglich im Kontext der objektorientierten Programmierung entwickelt wurden, wird beispielsweise in (Rogerson, 1997) deren Übertragbarkeit auf Software-Komponenten demonstriert. Wenn eine Software-Komponente nicht selbst über die Möglichkeiten zur Lösung einer Aufgabe verfügt, kann sie Nachrichten an andere Software-Komponenten verschicken und diese um Unterstützung bitten. Wenn die helfende Software-Komponente als Bestandteil der um Unterstützung ersuchenden Software-Komponente wirkt, wird dies als Komposition von Software-Komponenten bezeichnet. Die beteiligten Software-Komponenten werden auch äußere und innere Software-Komponente genannt. Eine Komposition von Software-Komponenten liegt dann vor, wenn die äußere Software-Komponente eine exklusive Referenz der inneren Software-Komponente hält. Die äußere Software-Komponente implementiert die benötigte Funktionalität nicht selbst, sondern nutzt die der inneren Software-Komponente. Sobald sich die Implementierung der inneren Software-Komponente ändert, bewirkt dies automatisch eine Änderung des Verhaltens der äußeren Software-Komponente.

Die Unterschiede zwischen Implementierungsvererbung und Komposition sind gering und sollen deshalb anhand eines Beispiels verdeutlicht werden.

In Abbildung 3.3 ist eine Klassenhierarchie dargestellt. Die Klasse CX erbt sämtliche Methoden und Eigenschaften der Klasse CY. Somit stellt die Menge der Methoden des Interfaces der Klasse CX eine Obermenge der Methoden des Interfaces der Klasse CY dar – Objekte der Klasse CX können folglich wie Objekte der Klasse CY behandelt werden. Die Funktionalität der Klasse CY kann durch die Definition weiterer Methoden und Eigenschaften in der Klasse CX ergänzt und spezialisiert werden. Gleichzeitig kann die Implementierung ausgewählter Methoden überschrieben werden.

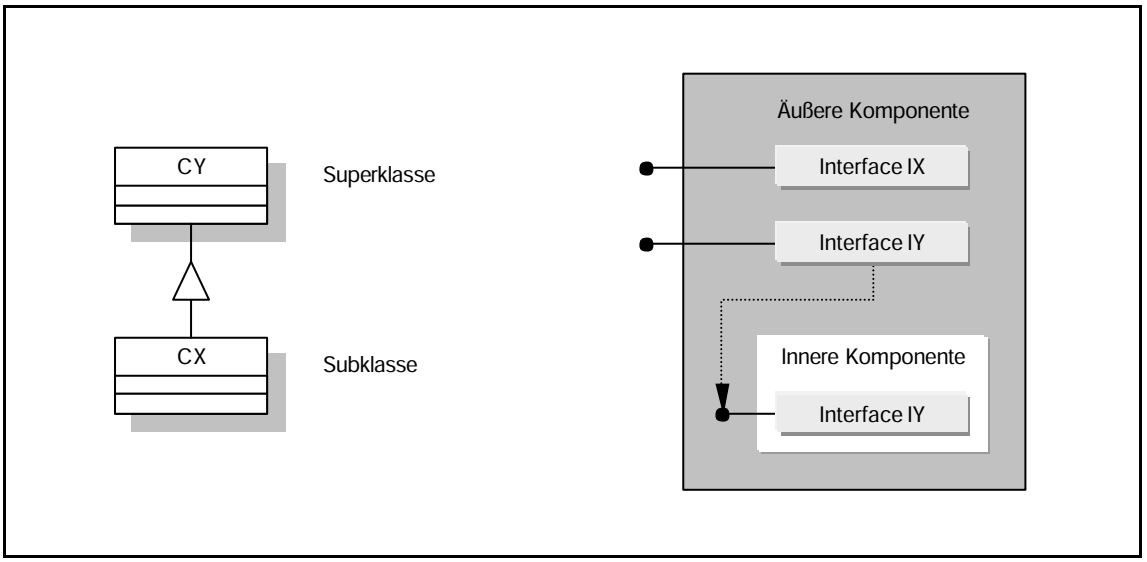


Abbildung 3.3: Wiederverwendung von Programm-Code durch Vererbung und Komposition

Ebenfalls in Abbildung 3.3 ist ein Beispiel für die Komposition von Software-Komponenten dargestellt. Die äußere Software-Komponente verfügt über ein eigenes Interface IY, leitet aber alle Methodenaufrufe an die innere Software-Komponente weiter. Dazu nutzt es über das Interface IY die Funktionalität der inneren Software-Komponente. Außerdem verfügt die äußere Software-Komponente über ein Interface IX und die zugehörige Implementierung. Ähnlich wie durch Vererbung kann damit die Funktionalität der inneren Software-Komponente ergänzt und spezialisiert werden.

Ein wesentlicher Unterschied zwischen Vererbung und Komposition liegt in der Art der Modellierung mit Relationen. Während eine Vererbungshierarchie durch die Vererbung von Interfaces "is a"-Relationen abbildet, stellen Kompositionen von Software-Komponenten mit Hilfe von Referenzen "has a"-Relationen dar. Da Software-Komponenten jedoch beliebige Interfaces implementieren können, ist ebenso die Modellierung von "is a"- Relationen möglich. Für die praktische Anwendung hat diese Unterscheidung keine gravierenden Auswirkungen, so dass in vielen Fällen das Konzept der Vererbung durch das Konzept der Komposition ersetzt werden kann.

Kapitel 4
Basistechnologien für Komponenten-Software

Im vorangegangenen Kapitel wurden die Grundlagen der komponentenorientierten Software-Technik auf einer abstrakten Ebene diskutiert, ohne dabei auf konkrete Technologien zur Realisierung einzugehen. In diesem Kapitel werden Merkmale und Besonderheiten der konkurrierenden Basistechnologien gegenübergestellt und bewertet. Das Ziel des Kapitels ist es, eine möglichst objektive Entscheidungsgrundlage zu bieten, welche Standards für Komponentenmodelle und Verbunddokumente als Plattform ausgewählt werden sollten. Weiterhin werden gängige Programmiersprachen im Hinblick auf ihre Eignung zur Erstellung und Einbindung von Software-Komponenten untersucht und bewertet.

4.1 Standards für Komponentenmodelle

Damit unabhängig voneinander entwickelte Software-Komponenten zusammenarbeiten können, wird ein Komponentenmodell benötigt, das die Regeln definiert, durch die Software-Komponenten auf binärer Ebene kompatibel gemacht werden können. Dazu ist nach (Pfister, 1997) unter anderem eine Standardisierung bezüglich der folgenden Punkte erforderlich:

- Interface Definition Language
- Interface Repository
- Implementation Repository
- Versionierung
- Calling Conventions
- Polymorphes Binden
- Speicherverwaltung
- verteilte Software-Komponenten

Damit Software-Entwickler auf vorgefertigte Software-Komponenten zurückgreifen können, müssen deren Interfaces in einer für Menschen lesbaren Form beschrieben werden. Zu diesem Zweck kann entweder eine spezielle Interface Definition Language (IDL) oder idealerweise eine Teilmenge des Sprachumfangs der zur Entwicklung eingesetzten Programmiersprache benutzt werden.

Zusätzlich wird auch eine binäre Beschreibung des Interfaces benötigt, die es dem Compiler ermöglicht, die syntaktisch korrekte Benutzung des Interface sicherzustellen. Eine Sammlung der von Software-Komponenten angebotenen Interfaces wird als Interface Repository bezeichnet. Da die textuelle und die binäre Beschreibung eines Interfaces die gleichen Informationen enthalten, kann das Interface Repository automatisch erzeugt werden.

Eine Software-Komponente wird erst dann geladen und dynamisch gebunden, wenn eine ihrer Methoden aufgerufen wurde. Zu diesem Zweck muss die das Interface implementierende Software-Komponente zur Laufzeit lokalisiert werden. Die dazu benötigten Informationen werden in einem Implementation Repository verwaltet.

Das Problem der Versionierung von Interfaces von verteilt entwickelten Software-Komponenten konnte bislang nicht zufriedenstellend gelöst werden. Deshalb sollten Interfaces und deren Spezifikation nach ihrer Publikation wie in Kapitel 3.3.2 beschrieben als unveränderlich angesehen werden.

Nachdem eine Software-Komponente lokalisiert, geladen und gebunden wurde, können deren Methoden aufgerufen werden. Im Regelfall werden dabei die Calling Conventions des jeweiligen Betriebssystems respektiert. Dadurch können alle diesen Konventionen folgenden Programmiersprachen die Software-Komponente ansprechen. Auf diesem Weg kann eine Sprachunabhängigkeit von Software-Komponenten erreicht werden.

Um die durch Software-Komponenten ermöglichte Vielfalt an Implementierungen optimal ausnutzen zu können, wird ein generischer Algorithmus benötigt, der es erlaubt, die Funktionalität eines Interfaces dynamisch zur Laufzeit zu untersuchen und polymorphe Implementierungen zu binden.

In einem System von Software-Komponenten kommt dem Speichermanagement zentrale Bedeutung zu. Die Bestimmung der erforderlichen Lebensdauer einer Software-Komponente ist nicht trivial, da diese durch eine Vielzahl anderer Software-Komponenten referenziert worden sein kann. Ein Lösungsansatz besteht darin, dass jede Software-Komponente über einen Referenzzähler verfügt. Für jede neue oder nicht mehr benötigte Referenz wird dieser Zähler inkrementiert oder dekrementiert. Sobald der Referenzzähler auf Null sinkt, wird die Software-Komponente nicht mehr benötigt und kann entladen werden. Ein versehentliches zu frühes Entladen kann durch diesen Mechanismus vermieden werden, und somit auch die Existenz ungültiger Pointer. Nicht ausgeschlossen werden kann jedoch das Auftreten von sogenannten Memory Leaks, da Software-Komponenten durch gegenseitige Referenzen ihr Entladen verhindern können. Ein als Garbage Collection bezeichnetes automatisches Speichermanagement löst dieses Problem und stellt deshalb ein extrem wertvolles Hilfsmittel dar.

Sollen Software-Komponenten auch in einer Netzwerkumgebung funktionsfähig sein, muss vom jeweiligen Komponentenmodell eine geeignete Infrastruktur zur Verfügung gestellt werden. Gewöhnlich werden zu diesem Zweck sogenannte Proxy-Objekte benutzt. Die Software-Komponente kommuniziert dabei mit dem lokalen Proxy-Objekt. Dieses linearisiert die Nachricht in eine binäre Zeichenfolge und leitet sie über das Netzwerk an das zweite Proxy-Objekt weiter. Dort wird die Nachricht wieder delinearisiert. Anschließend ruft das zweite Proxy-Objekt die gewünschte Software-Komponente auf. Die Rückgabewerte werden nach dem gleichen Verfahren übertragen. Der beschriebene Vorgang des Informationsaustausches wird auch als Marshaling bezeichnet. Für den Nutzer einer Software-Komponente darf dabei nicht erkenntlich sein, ob sich diese auf dem lokalen oder einem anderen Rechner befindet.

Im Folgenden werden die drei wichtigsten Standards für Komponentenmodelle vorgestellt und im Anschluss einer Bewertung unterzogen.

4.1.1 CORBA

Die Spezifikation der Common Object Request Broker Architecture (CORBA) wird durch die Object Management Group (OMG) entwickelt (OMG, 2000). Die OMG wurde 1989 gegründet und stellt mit mehreren Hunderten Mitgliedsfirmen das mit Abstand größte Konsortium der Computerindustrie dar.

Die Entwicklung von CORBA soll eine Integration von Software-Produkten ermöglichen, unabhängig von der verwendeten Programmiersprache und der verwendeten Plattform. Zu diesem Zweck standardisiert die OMG in sehr allgemeiner Form die Architektur eines Komponentenmodells. Die Kompatibilität der einzelnen Software-Produkte wird auf der Ebene des Quell-Codes und durch standardisierte Nachrichtenformate erreicht. Der offene Ansatz ermöglicht es einer Vielzahl von Software-Herstellern entsprechend ihrer konkreten Erfordernisse individuelle Lösungen anzubieten. Auf der anderen Seite bewirkt die fehlende Standardisierung auf binärer Ebene, dass Interoperabilität nur über die Anwendung aufwändiger High-Level-Protokolle erreicht werden kann.

Praktisch haben sich mehrere konkurrierende Implementierungen des Standards wie IONAs Orbix, Visigenics Visibroker und IBMs SOM etabliert. Im Verbund mit individuellen Gestaltungsspielräumen ist dadurch eine generelle Interoperabilität von Software-Komponenten nicht gegeben. In einigen Fällen ist eine Integration auf der Ebene des Quell-Codes nicht praktikabel oder wird aus Wettbewerbsgründen nicht akzeptiert.

Um dennoch eine Interoperabilität von mehreren Objekten zu erreichen, ist die Einhaltung zahlreicher Konventionen erforderlich. Folglich werden Client und Object Implementation häufig durch dasselbe Team entwickelt, auch wenn sie sich auf verschiedenen Systemen befinden und in unterschiedlichen Programmiersprachen implementiert werden.

Bislang wurde die Arbeit der Entwickler nur durch grundlegende Mechanismen zur Unterstützung netzweiter Anfragen unterstützt. Diesem Mangel versucht die OMG durch die Entwicklung zusätzlicher Dienste (CORBAservices) und Infrastruktur (CORBAfacilities) zu begegnen. Diese erweiterten Aktivitäten werden unter dem Begriff Object Management Architecture (OMA) zusammengefasst.

Die Architektur von CORBA orientiert sich stark am herkömmlichen objektorientierten Ansatz, so dass jedes Objekt über genau ein Interface verfügt. Das Interface wird durch OMG IDL spezifiziert und in einem Interface Repository hinterlegt. Geeignete Implementierungen können über das Implementation Repository lokalisiert werden. Bezüglich der verwendeten Calling Conventions, des polymorphen Verhaltens von Software-Komponenten, des Speicher-Managements werden keine allgemein gültigen Regelungen getroffen.

Die Verwendung verteilter Software-Komponenten wird durch den Object Request Broker (ORB) vereinfacht. Die Nachrichtenübergabe erfolgt durch Remote Procedure Calls. Die dazu benötigten Paare von Stubs und Skeletons werden aus OMG-IDL erzeugt. Über den Object Adapter kann das Objekt mit dem ORB kommunizieren. Theoretisch ist die Entwicklung spezialisierter Object Adapter möglich. Bislang wurde nur der Basic Object Adapter (BOA) standardisiert, der für die Kommunikation mit wenigen großen Objekten optimiert ist.

In Abbildung 4.1 ist vereinfacht der Ablauf einer Anfrage des Clients an ein Objekt dargestellt. Der Client stellt eine Anfrage entweder über das Invocation Interface oder über ein Stub. Aufgabe des ORB ist es, anhand des Implementation Repository die gewünschte Object Implementation zu lokalisieren. Anschließend übermittelt er die übergebenen Parameter und überträgt dem Objekt die Kontrolle. Das Objekt kann über den Object Adapter Dienste des ORB anfordern. Nachdem die Anfrage des Clients abgearbeitet wurde, transferiert der ORB die Rückgabewerte an den Client.

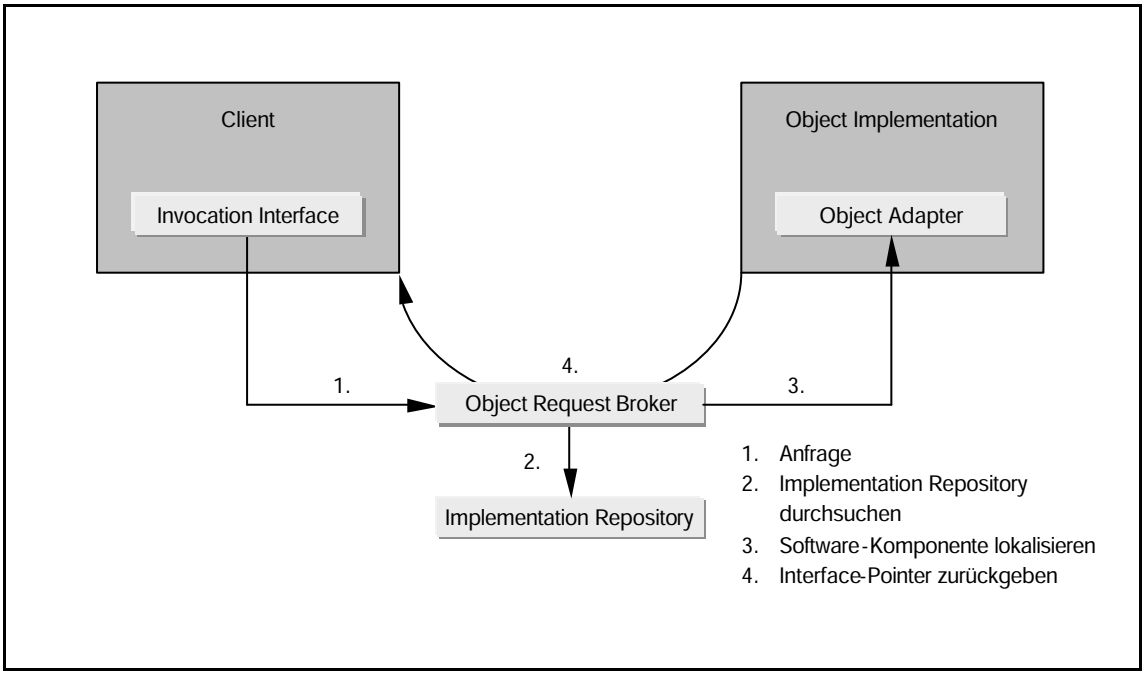


Abbildung 4.1: Lokalisieren und Aktivieren eines Objektes mit CORBA

4.1.2 COM

Das Component Object Model wurde von der Firma Microsoft entwickelt. Die erste Anwendung war die ab 1993 verfügbare und als OLE2 bezeichnete verbesserte Funktionalität für Verbunddokumente. Mittlerweile basieren alle Betriebssysteme und Applikationen von Microsoft auf COM. Infolgedessen stellt COM den am weitesten verbreiteten Standard für ein Komponentenmodell dar. Detaillierte Informationen können beispielsweise (Microsoft, 1995a), (Brockschmidt, 1995), (Rogerson, 1997) und (Box, 1997) entnommen werden.

Im Gegensatz zu CORBA oder Java kann eine COM-Komponente aus mehreren Objekten und mehreren Interfaces bestehen. Zur Definition von Interfaces können wahlweise Mittel der jeweiligen Programmiersprache oder eine als Microsoft IDL bezeichnete Erweiterung von DCE[1]-IDL benutzt werden. Aus diesen Informationen können sogenannte Type Libraries als binäre Beschreibungen der Interfaces erzeugt werden. Die verfügbaren COM-Komponenten werden in der als Implementation Repository fungierenden Registry verwaltet.

[1] Distributed Computing Environment

Zur Identifizierung von COM-Komponenten, Interfaces und zu anderen Zwecken verwendet COM sogenannte Globally Unique Identifiers (GUID). Ein GUID ist ein 128 bit langer Bezeichner, der wiederum von den DCE Universally Unique Identifiers (UUID) abgeleitet ist. Ein GUID wird unter Verwendung der IP-Adresse des Rechners, der Systemuhrzeit und einer Zufallszahl erzeugt, so dass dieser in Zeit und Raum eindeutig ist. GUIDs zur Identifizierung von COM-Komponenten werden als Class Identifier (CLSID) bezeichnet, wohingegen Interfaces durch Interface Identifiers (IID) gekennzeichnet werden.

In Abbildung 4.2 ist vereinfacht der Algorithmus zum Lokalisieren und Aktivieren von COM-Komponenten dargestellt. Wenn eine COM-Komponente die Dienste einer anderen COM-Komponente in Anspruch nehmen möchte, sendet sie eine Anforderung an COM, die durch die übergebene CLSID repräsentierte COM-Komponente zu lokalisieren und erforderlichenfalls zu instanziieren. Anschließend gibt COM einen Zeiger auf das durch die ebenfalls übergebene IID spezifizierte Interface zurück. Danach kann die Kommunikation auf direktem Weg, das heißt ohne Mitwirkung von COM, erfolgen. Da eine COM-Komponente mehrere Klassen implementieren kann und in denselben oder einen getrennten Adressraum geladen werden kann, werden zu deren Erzeugung spezielle Factory Objects benötigt. Diese werden unter COM jedoch Class Factories genannt.

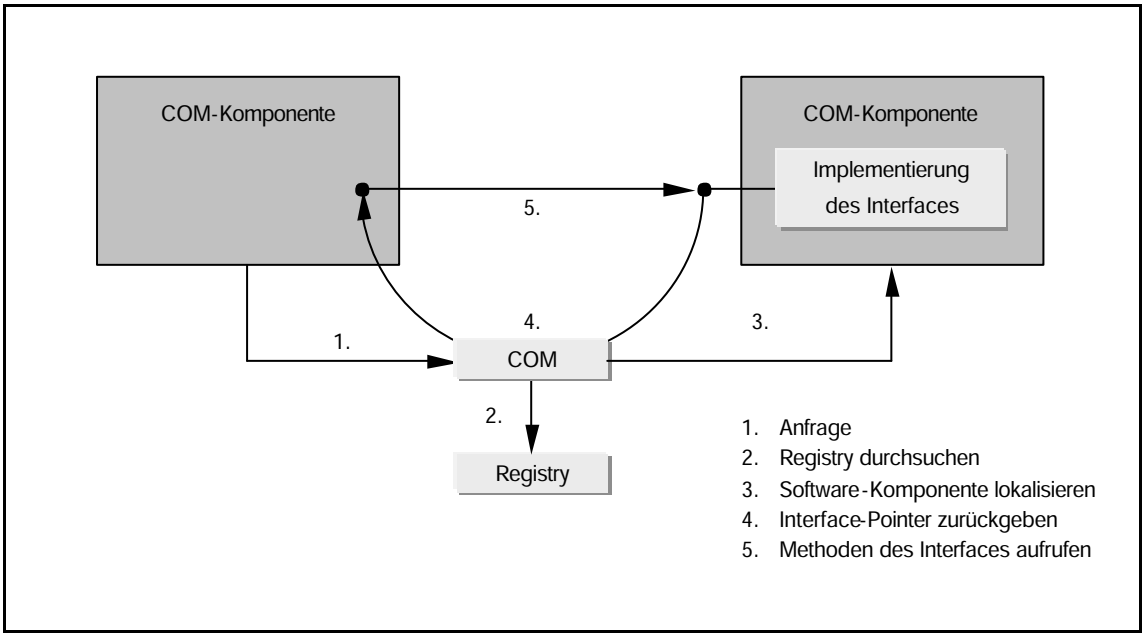


Abbildung 4.2: Lokalisieren und Aktivieren einer Software-Komponente mit COM

Indem COM sämtliche Interfaces ab dem Zeitpunkt ihrer Publikation als unveränderlich deklariert, tritt sowohl das Problem der Versionierung als auch das Fragile-Base-Class-Problem nicht in Erscheinung. Mehrere Versionen können unterstützt werden, da jede COM-Komponente eine beliebige Zahl von Interfaces implementieren kann. Mehrere Versionen desselben Interfaces werden folglich als vollkommen unabhängige Interfaces unterstützt.

Im Unterschied zu CORBA stellt COM einen binären Standard dar, der lediglich Calling Conventions und ein einheitliches binäres Speicher-Layout vereinbart. Auf der Ebene des Quell-Codes werden keinerlei Vereinbarungen getroffen. Das wesentliche durch COM definierte Element ist das Interface. Wie in Abbildung 4.3 dargestellt, wird ein Interface durch einen Zeiger repräsentiert, der seinerseits auf die sogenannte Virtual Function Table (vtable) verweist. Deren Einträge verweisen auf die Implementierungen der einzelnen Methoden.

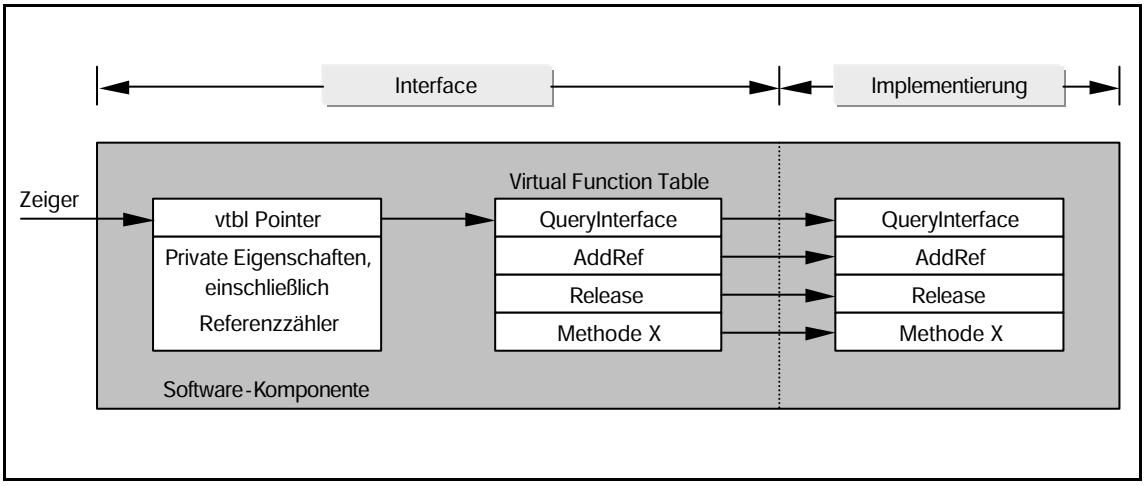


Abbildung 4.3: Binäres Speicher-Layout einer COM-Komponente

Der Typ einer COM-Komponente wird nicht durch Interface-Vererbung, sondern durch Implementierung beliebiger, voneinander völlig unabhängiger Interfaces definiert. Jedes Interface wird vom Basis-Interface IUnknown abgeleitet und verfügt dadurch über dessen Methoden QueryInterface, AddRef und Release. Mit Hilfe der Methode QueryInterface kann festgestellt werden, ob die COM-Komponente über das als Parameter übergebene Interface verfügt. Gegebenenfalls wird eine Referenz auf das spezifizierte Interface zurückgegeben. Ob eine COM-Komponente dem benötigten Typ entspricht, kann somit über eine Folge von Aufrufen der Methode QueryInterface ermittelt werden. Die Instanziierung und eine möglicherweise große Anzahl von Aufrufen der Methode QueryInterface stellt eine sehr ineffiziente Möglichkeit zur Bestimmung des Typs einer COM-Komponente dar. Aus diesem Grund gestattet COM die Definition sogenannter Kategorien. Kategorien werden durch einen Category Identifier (CATID) genannten GUID gekennzeichnet und durch die Registry verwaltet. Ist eine COM-Komponente in einer solchen Kategorie eingetragen, muss sie alle vereinbarten Interfaces implementieren.

Da COM keine Implementierungsvererbung unterstützt, werden für die Wiederverwendung durch Komposition (vergleiche Kapitel 3.3.3) die Mechanismen Containment und Aggregation unterstützt (Abbildung 4.4). Während Containment den Standardfall für die Komposition von Software-Komponenten darstellt, vermeidet Aggregation den Aufwand für die Weiterleitung der Methodenaufrufe. Da Aggregation im Gegensatz zum Containment jedoch eine Kooperation der zu aggregierenden Software-Komponente erfordert, erhöht sich die Komplexität beträchtlich. Aggregation wird deshalb nur selten angewendet und ist nur zur Optimierung der Performance sinnvoll.

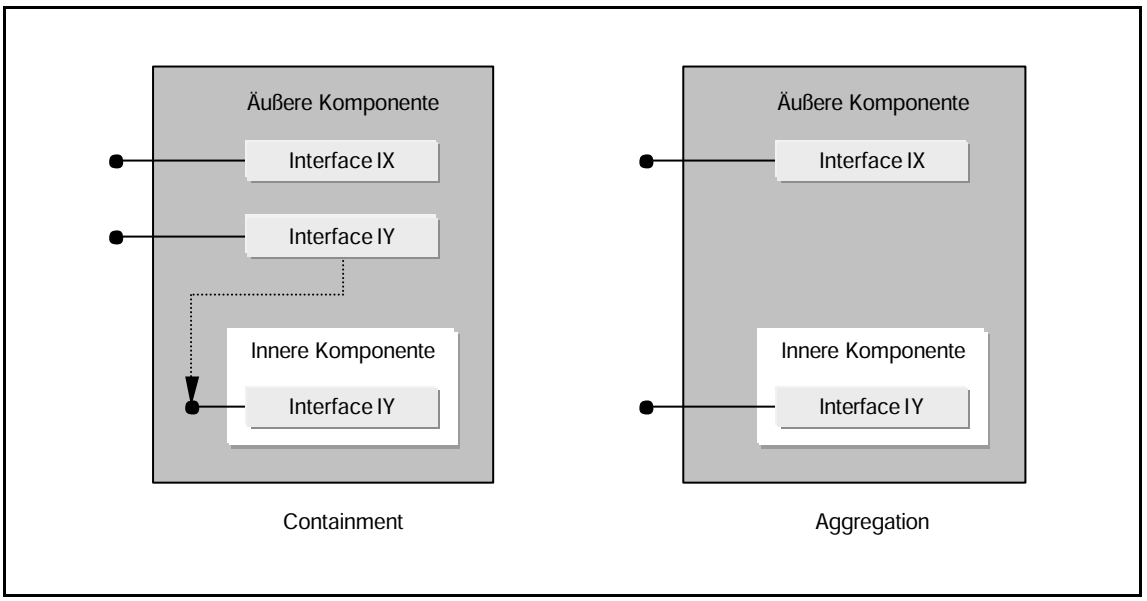


Abbildung 4.4: Wiederverwendung von Programm-Code durch Komposition (Containment und Aggregation)

Das Speichermanagement erfolgt unter COM durch Referenzzählung. Jede COM-Komponente zählt die auf sie verweisenden Referenzen. Für jede neu erzeugte Referenz oder nicht mehr benötigte Referenz wird durch Aufruf der Methoden AddRef oder Release der Referenzzähler inkrementiert beziehungsweise dekrementiert. Ein Referenzzähler mit dem Wert Null signalisiert, dass eine Software-Komponente nicht mehr referenziert wird und somit entladen werden kann.

Die einfachste und effizienteste Form einer COM-Komponente wird durch eine In-Process-Komponente realisiert. In-Process-Komponenten werden als DLL in den gemeinsamen Adressraum geladen. Wenn Sicherheit von vorrangigem Interesse ist, können COM-Komponenten auch als EXE-Dateien zu Out-of-Process-Komponenten gebunden werden. Da diese in einem separaten Adressraum ausgeführt werden, ist das Laufzeitverhalten wesentlich schlechter, da Daten nur über den Mechanismus zur Inter-Process-Communication ausgetauscht werden können. Es werden spezielle Proxy- und Stub-Module benötigt, welche die Linearisierung und Delinearisierung der Daten übernehmen. In einer Distributed COM (DCOM) genannten Erweiterung von COM werden Proxy und Stub benutzt, um über Remote Procedure Calls (RPC) verteilte COM-Komponenten anzusprechen.

Neben der ursprünglichen Windows-Plattform waren COM-Implementierungen auch für andere Plattformen wie Apples MacOS und Unix-Derivate geplant.

Die Entwicklung von COM-Komponenten erfordert – zumindest mit der Sprache C++ – einen großen Entwicklungsaufwand. Durch eine COM+ genannte Erweiterung von COM sollen zukünftig Standardaufgaben durch eine COM+-Runtime und durch COM+-Services adressiert werden. Beispielsweise ist vorgesehen, dass die Interface-Definition durch Mittel der verwendeten Programmiersprachen erfolgen kann und dass eine auf Garbage Collection beruhende automatische Speicherverwaltung angeboten wird.

4.1.3 Java

Im Unterschied zu den sprachneutralen Komponentenmodellen von COM und CORBA handelt es sich bei Java um ein sprachspezifisches Komponentenmodell. Java wurde durch Sun Microsystems entwickelt und ist seit 1995 verfügbar. Java wurde primär für die Entwicklung von Applets und nicht im Hinblick auf die für Software-Komponenten relevanten Gesichtspunkte entworfen. Applets stellen Mini-Applikationen dar, die direkt im Adressraum eines WWW-Browsers ausgeführt werden können. Da die Sicherheit von über das Internet bezogener Software nicht generell gewährleistet werden kann, stellt der Schutz vor fehlerhafter oder zerstörerischer Software ein vordergründiges Anliegen dar. Dementsprechend war größtmögliche Sicherheit ein wesentliches Entwicklungsziel. Daraus resultiert auch die Forderung nach Typsicherheit, deren Erfüllung die Entwicklung einer neuen Programmiersprache nötig gemacht hat, da durch gängige Sprachen wie C++ dieses Kriterium nicht erfüllt werden konnte.

Obwohl es ohne weiteres möglich ist, mehrere Java-Applets auf einer HTML-Seite zu platzieren, können diese nicht direkt miteinander kommunizieren, da sie über keine entsprechende Infrastruktur verfügen. Stattdessen ist eine Interaktion generell nur über Server möglich. Java-Applets stellen somit im Sinne der Definition in Kapitel 3.1.1 keine Software-Komponenten dar. Durch die Entwicklung der JavaBeans und zusätzlicher Dienste (zum Beispiel Swing) werden jedoch die Voraussetzungen für die Entwicklung von Software-Komponenten bereitgestellt werden.

Ein herausragendes Merkmal von Java ist die Möglichkeit der Plattformunabhängigkeit. Java Quell-Code wird durch den Compiler in einen plattformunabhängigen Java Byte-Code übersetzt und in speziellen Class-Dateien gespeichert. Zur Laufzeit wird der Byte-Code durch die Java Virtual Machine (JVM) und einen Just-in-Time-Compiler in das native Format der jeweiligen Plattform übersetzt. Durch Implementierung der JVM auf allen relevanten Plattformen wird eine weitgehende Portabilität erreicht. Theoretisch ist es auch möglich, dass andere typsichere Sprachen wie Visual Basic oder Smalltalk Binärcode für die JVM erzeugen.

Der größte Vorteil des sprachspezifischen Komponentenmodells von Java liegt in der Verwendung einer Teilmenge des Sprachumfangs als Interface Definition Language. Aus der Sicht des Programmierers stellt dies die am nächsten liegende und einfachste Möglichkeit zur Interface-Definition dar. Zu diesem Zweck unterscheidet Java zwischen Klassen und Interfaces. Ein Objekt wird dabei durch seine Klasse definiert, kann aber über mehrere Interfaces verfügen. Alternativ dazu können Interfaces auch durch OMG-IDL definiert werden.

Ein spezielles Interface Repository wird nicht benötigt, da alle Informationen direkt aus den standardisierten Class-Dateien ausgelesen werden können. Ein Implementation Repository ist ebenfalls nicht vorhanden, stattdessen werden reguläre Dateinamen benutzt, die über einen standardisierten Verzeichnisbaum identifiziert werden. Eine einfache Versionsprüfung wird zur Laufzeit durchgeführt. Bezüglich des Speicher-Layouts werden keine Regelungen getroffen, stattdessen definiert Java ein einheitliches Dateiformat.

Das Java Native Interface (JNI) spezifiziert für alle Plattformen die nativen Calling Conventions. Somit kann auch Programm-Code außerhalb des Browsers aufgerufen werden. Der Zugriff auf externe Objekte erfolgt über JNI Interface Pointer, die sich in ihrer Struktur sehr stark an COM-Interfaces orientieren.

Aufgrund der erhöhten Sicherheitsanforderungen sind einfache Mechanismen zur Referenzzählung nicht ausreichend. Stattdessen kommt ein automatisches Speichermanagement zum Einsatz. Der als Garbage Collection bezeichnete Mechanismus entfernt selbständig nicht mehr benötigte Objekte aus dem Speicher. Intern erfolgt weiterhin eine Referenzzählung. Da diese jedoch nicht mehr im Verantwortungsbereich des Programmierers liegt, wird dieser vom zugehörigen Verwaltungsaufwand befreit. Gleichzeitig werden Fehlerkategorien wie ungültige Zeiger oder Memory-Leaks infolge zyklischer Referenzen ausgeschlossen.

Verteilte Software-Komponenten werden durch das Java Remote Method Invocation Interface (RMI) unterstützt. Zu diesem Zweck wird ein im Vergleich zu den konkurrierenden Komponentenmodellen herausragendes Merkmal realisiert, indem das Speichermanagement um Distributed Garbage Collection erweitert wird. Alternativ können auch die Mechanismen von COM und CORBA benutzt werden, um verteilte Software-Komponenten zu realisieren.

4.1.4 Vergleich und Bewertung

Die grundlegenden Konzepte komponentenorientierter Software-Technik wie dynamische Polymorphie, Kapselung und spätes Binden werden von allen untersuchten Komponentenmodellen unterstützt. Dennoch bestehen eine Reihe fundamentaler Unterschiede. Die wesentlichen Aspekte sind in Tabelle 4.1 gegenübergestellt.

Im Unterschied zu COM und Java basiert CORBA nicht auf binären Software-Komponenten. Darüber hinaus ist durch die konkurrierende Vielfalt der CORBA-Implementierungen und der Notwendigkeit sprachspezifischer Lösungen eine generelle Kompatibilität der Software-Komponenten nicht gegeben. Infolgedessen ist CORBA insbesondere zur Realisierung komplexer Unternehmenslösungen geeignet.

COM erfüllt alle wesentlichen Anforderungen an ein Komponentenmodell. Zahlreiche Detaillösungen wie Unveränderlichkeit von Interfaces, eindeutige Identifizierung durch GUIDs, Kategorien und Sprachneutralität stellen sehr zweckmäßige Regelungen dar. Dennoch ist die praktische Anwendung bislang sehr aufwändig. Dieser Mangel soll durch COM+ behoben werden. Für Anwendungsgebiete, die eine große Zahl von Software-Komponenten erfordern, und zur Einbindung von Standard-Software der Windows-Plattform stellt COM das gegenwärtig am besten geeignete Komponentenmodell dar.

Die Entwicklung von Java nähert sich in vielen Punkten an den von COM beschrittenen Weg an und bietet vereinzelt innovativere Konzepte. Allerdings ist Java vordergründig auf Internet-Umgebungen optimiert und noch nicht genügend ausgereift. Die Entwicklung von komplexen Software-Komponenten wird bislang nicht ausreichend unterstützt. Langfristig besteht die Möglichkeit, dass sich Java zum stärksten Konkurrenten von COM entwickelt.

	Corba	COM	Java
Urheber/ Eigentümer	OMG	Microsoft/Open Group	Sun Microsystems
Interface Definition	OMG-IDL	Microsoft IDL	Java Syntax
Interface Repository	ja	ja	ja
Implementation Repository	ja	ja	ja
Versionsprüfung	nein	entfällt	ja
Binärer Speicherstandard	nein	ja	nein
Binärer Dateistandard	nein	nein	ja
Interface-Vererbung	ja	ja	ja
Implementierungsvererbung	nein	nein	einfach
Speicherverwaltung	nein	Referenzzählung	Garbage Collection
Verteilte Umgebungen	ja	DCOM	Java RMI
Sprachunabhängigkeit	ja	ja	nein
Plattformunabhängigkeit	ja	nein	ja

Tabelle 4.1: Standards für Komponentenmodelle

4.2 Standards für Verbunddokumente

Herkömmliche monolithische Applikationen benutzen im Regelfall proprietäre Verfahren, um Informationen darzustellen und zu speichern. Die Übertragung dieses Ansatzes auf Software-Komponenten würde bedeuten, dass jede Software-Komponente für die Darstellung und Speicherung ihrer Informationen verantwortlich ist. Die dadurch bewirkte Fragmentierung von Darstellung und Daten ist weder zweckmäßig noch praktikabel.

Komponentenbasierte Software verdeutlicht, dass es aus der Perspektive des Nutzers nebensächlich ist, welche Applikation oder Software-Komponenten zur Bearbeitung eines Problems benötigt werden. Wesentlich für den Nutzer ist hingegen, wie er bei der Lösung eines Problems und dessen Dokumentation unterstützt wird. Offensichtlich ist es deshalb in vielen Fällen zweckmäßig, Daten und deren Darstellung unabhängig von den zur Bearbeitung benutzen Software-Komponenten zu problemspezifischen Dokumenten zusammenzufassen. Derartige Dokumente werden als Verbunddokumente bezeichnet. Für Verbunddokumente wird eine über Komponentenmodelle hinausgehende Infrastruktur benötigt. Der Schwerpunkt der benötigten Standardisierung liegt auf der Teilung der folgenden Ressourcen:

- Darstellung
- Persistenz
- Scripting

Die dazu erforderliche Kooperation der Software-Komponenten muss durch Regeln definiert werden. Die Gesamtheit dieser Regeln bildet einen Vertrag, der praktisch durch eine Menge von Interfaces beschrieben wird.

4.2.1 OLE

OLE wurde erstmalig 1991 als Bestandteil von Windows 3.1 ausgeliefert. 1993 erfolgte eine komplette Überarbeitung, wobei ein Wechsel der Basistechnologie von Dynamic Data Exchange (DDE) zu COM erfolgte. Seitdem stellt OLE den Standard für Verbunddokumente der Windows-Plattform dar.

Vordergründig definiert OLE eine Architektur für die gemeinsame visuelle Darstellung der Daten von Software-Komponenten in einem gemeinsamen Dokument. Die Darstellung kann beliebige polygonale, sich überlappende Bereiche umfassen. Um dem Nutzer die Illusion zu geben, mit einem einzigen Dokument zu arbeiten, wird als besonderes Merkmal die sogenannte In-Place-Activation unterstützt. Entsprechend den Erfordernissen der zur Bearbeitung benutzten Software-Komponente können Menüs und Symbolleisten angepasst werden.

Darüber hinaus stellt OLE weitere Dienste zur Verfügung. So können mit Structured Storage Dateien hierarchisch strukturiert werden. Mit Hilfe dieses Dateisystems innerhalb einer Datei können die persistenten Daten von Software-Komponenten zu logischen Einheiten zusammengefasst werden.

Software-Komponenten mit einer auf OLE basierenden Nutzeroberfläche werden als ActiveX-Controls bezeichnet. Diese Controls können über einen Automation genannten Scripting Dienst gesteuert werden. Da eine programmatische Steuerung der Controls ebenso über den wesentlich effizienteren COM-Mechanismus erfolgen kann, kommt Automation nur noch geringe Bedeutung zu.

Bedingt durch den evolutionären Entwicklungsprozess wurde eine Vielzahl von Interfaces definiert, die teilweise redundant oder inkonsistent sind. Dadurch wird die Entwicklung beträchtlich erschwert. Durch den pragmatischen Ansatz können andererseits auch herkömmliche Applikationen unterstützt werden. Der somit ermöglichte allmähliche Übergang zu komponentenbasierten Applikationen stellt eine wesentliche Ursache für den Erfolg von OLE dar.

4.2.2 OpenDoc

Als Alternative zu OLE wurde 1993 durch Apple, IBM und zahlreiche Partner die Entwicklung von OpenDoc als eigene Technologie für Verbunddokumente angekündigt. Zur Entwicklung wurde ein Component Integration Labs (CILabs) genanntes Konsortium gegründet.

Obwohl die Entwicklung 1997 eingestellt wurde, stellt OpenDoc einen erwähnenswerten Ansatz dar. Zum einen entfielen durch die komplette Neuentwicklung durch Kompatibilität bedingte Restriktionen. Infolgedessen konnte die Architektur von OpenDoc wesentlich einfacher, klarer und mit durchgängiger Konsistenz gestaltet werden. Vergleichbare Aufgaben können mit weniger Interfaces mit einer geringeren Anzahl von Methoden als unter OLE realisiert werden. Zum anderen stellt OpenDoc im Gegensatz zu OLE und JavaBeans ein Component Framework (vergleiche Kapitel 5.3.2.2) dar, das die Konformität der installierten Software-Komponenten erzwingt. Im Unterschied dazu sind die Regeln für ActiveX-Controls lediglich durch eine Spezifikation definiert.

OpenDoc verwendet als Komponentenmodell IBMs System Object Model SOM. Die Schwächen des benutzten Komponentenmodells müssen durch OpenDoc ausgeglichen werden. Da SOM kein Verfahren zum Speichermanagement anbietet, musste ein eigener Mechanismus zur Referenzzählung implementiert werden. Außerdem wurde Funktionalität zur dynamischen Analyse von Interfaces benötigt. Dazu werden die Methoden HasExtension und GetExtension benutzt, die vergleichbar mit COMs QueryInterface sind.

OpenDoc bietet im Wesentlichen dieselben Basisdienste wie OLE an. So können die LiveObjects genannten Software-Komponenten beliebige polygonale, sich überlappende Bereiche umfassen. Eine Anpassung der Menüs wird ebenso unterstützt.

Zur Speicherung der persistenten Daten verwendet OpenDoc Structured Files, die unter dem Namen Bento bekannt sind. Diese können auch mehrere Versionen eines Dokuments enthalten.

OpenDoc bietet ebenfalls ein als Open Scripting Architecture (OSA) bezeichnetes Scripting-Interface. Dieses stellt wie Automation unter OLE einen aufwändig zu implementierenden und redundanten Mechanismus dar.

4.2.3 JavaBeans

Die Entwicklung portabler Software-Komponenten auf der Basis von Java soll durch die seit 1996 verfügbaren JavaBeans ermöglicht werden. Der Schwerpunkt liegt dabei auf Controls von geringer bis mittlerer Komplexität. Derartige Controls sind zur Ausführung in einem Browser bestimmt. Zur Laufzeit können Controls weder hinzugefügt oder gelöscht werden noch kann ihre Größe verändert werden. Die Entwicklung von Containern wird nicht unterstützt. Die JavaBeans stellen somit keine vollständige Architektur für Verbunddokumente dar.

Die JavaBeans bieten lediglich eingeschränkte Möglichkeiten zur visuellen Repräsentation von Software-Komponenten. Nichtrechteckige oder überlappende Darstellungen sind nicht möglich, die Modifikation von Menüs soll erst durch zukünftige Versionen unterstützt werden. Eine persistente Speicherung von Daten ist nur bei Einhaltung der Sicherheitsvorschriften zulässig. Im Regelfall soll zu diesem Zweck der Browser benutzt werden. JavaBeans benötigen keinen speziellen Scripting-Mechanismus, sondern können durch Java oder JavaScript-Befehle gesteuert werden.

4.2.4 Vergleich und Bewertung

Alle drei Standards stellen Mechanismen zur visuellen Darstellung von Software-Komponenten bereit. Die wesentlichen Aspekte sind in Tabelle 4.2 gegenübergestellt. OLE ist seit langer Zeit am Markt etabliert, leidet aber unter der durch Kompatibilitätserfordernisse bestimmten Architektur. Trotz technischer Überlegenheit musste die Entwicklung von OpenDoc wegen fehlender Akzeptanz am Markt eingestellt werden. JavaBeans sind bislang ausschließlich auf die Entwicklung von Software-Komponenten ohne Container-Funktionalität fokussiert und noch nicht annähernd ausgereift. Zum gegenwärtigen Zeitpunkt stellt somit OLE den einzig brauchbaren Standard für Verbunddokumente dar.

	OLE	OpenDoc	JavaBeans
Urheber	Microsoft	Apple	Sun Microsystems
Eigentümer	Open Group	IBM	Sun Microsystems
Name der Komponenten	ActiveX Object	LiveObject	Bean
Komponentenmodell	COM	SOM	Java
Menu Sharing	ja	ja	geplant
Überlappende Darstellung	ja	ja	nein
Polygonale Darstellung	ja	ja	nein
Verbunddateien	Structured Storage	Bento	ja
Scripting	Automation	OSA	Java

Tabelle 4.2: Standards für Verbunddokumente

4.3 Komponentenorientierte Programmiersprachen

Bislang gab es bei der Wahl der einzusetzenden Programmiersprache kaum Alternativen: Aus Gründen der Effizienz und wegen der Möglichkeit zur Nutzung von vorgefertigten Class Frameworks wie beispielsweise den MFC stellte C++ für viele Aufgaben die Sprache der Wahl dar. Prinzipiell wäre es zwar möglich gewesen, mehrere Programmiersprachen innerhalb eines Projektes einzusetzen. Allerdings hätten die einzelnen Programmfragmente dann in einem aufwändigen Prozess unter Beachtung der unterschiedlichen Calling Conventions zu einer Anwendung zusammengefügt werden müssen. Infolgedessen wurde der überwiegende Anteil professionell entwickelter Software komplett in C++ geschrieben.

Durch sprachneutrale Komponentenmodelle können Software-Komponenten auf einfache Weise miteinander kombiniert werden, unabhängig von der zur Entwicklung eingesetzten Programmiersprache. Da Software-Komponenten in binärer Form ausgeliefert werden, spielt die Wahl der Programmiersprache nur noch eine marginale Rolle. Folglich kann für jede Software-Komponente individuell festgelegt werden, welche Programmiersprache für die konkreten Erfordernisse am besten geeignet ist.

Die wesentlichen Anforderungen an Entwicklungssysteme für komponentenorientierte Software-Technik wurden bereits in Kapitel 3.2.2 formuliert.

4.3.1 C++

Die Sprache C++ bietet dem Entwickler maximale Freiheiten, so dass das ohnehin schon gute Laufzeitverhalten durch Optimierungen nochmals verbessert werden kann. Die dazu benutzten Mittel wie Zeiger und manuelles Speichermanagement bewirken jedoch auch, dass C++ nicht typsicher ist. Nachteilig ist weiterhin, dass C++ ein Mechanismus zur Kapselung auf der Ebene von Modulen fehlt und dass keine Unterscheidung zwischen der Vererbung von Implementierungen und Interfaces vorgenommen wird.

Die Entwicklung von Software-Komponenten unter COM erfordert einen sehr hohen Implementierungsaufwand, da alle Einzelheiten vom Entwickler vorzugeben sind. Bis zur Verfügbarkeit von COM+ kann dieser Aufwand durch Verwendung von Bibliotheken wie der Active Template Library ATL oder den MFC etwas reduziert werden. Zur Zeit wird die Entwicklung unter C++ durch den Einsatz von Software-Komponenten nicht vereinfacht, sondern stellt nochmals wesentlich höhere Anforderungen an den Entwickler.

4.3.2 Visual Basic

Größtmögliche Einfachheit war das oberste Ziel bei der Entwicklung von Visual Basic. Zu diesem Zweck wurden viele Details verborgen, die zur Implementierung von Standardaufgaben nicht bekannt sein müssen. Infolgedessen sind Rückschlüsse auf das Laufzeitverhalten und somit auch Optimierungen kaum möglich. Durch den Verzicht auf Zeiger und durch die automatische Speicherverwaltung wird die Forderung nach Typsicherheit erfüllt. Eine Kapselung auf Modulebene ist nicht möglich.

Visual Basic unterstützt nicht alle Merkmale der objektorientierten Programmierung und gilt deshalb nur als objektbasierte, nicht aber als objektorientierte Sprache im Sinne von Kapitel 3.2.1. Beispielsweise wird das Konzept der Implementierungsvererbung nicht unterstützt. Dies stellt jedoch keine gravierende Einschränkung für die Entwicklung von Software-Komponenten dar, da eine Klasse in Visual Basic beliebig viele COM-Interfaces implementieren kann und somit polymorphes Verhalten unterstützt. Die Wiederverwendung von Code wird durch die Komposition von Software-Komponenten unterstützt. Dabei ist die Komplexität von COM für den Entwickler nicht sichtbar. Alle von Visual Basic benutzten ActiveX-Controls sind selbst Software-Komponenten. Genauso intuitiv ist auch die Arbeit mit anderen Software-Komponenten möglich. Infolgedessen stellen ActiveX-Controls den bislang erfolgreichsten Markt für Software-Komponenten dar.

4.3.3 Java

Die Sprache Java orientiert sich an dem von C++ gebotenen Sprachschatz. Oberstes Entwicklungsziel war größtmögliche Sicherheit. Die geforderte Typsicherheit konnte durch den Verzicht auf Zeiger und durch ein automatisches Speichermanagement erreicht werden. Durch sogenannte Packages ist die Modulsicherheit gewährleistet. Darüber hinaus wird sauber zwischen der Implementierungs- und Interface-Vererbung unterschieden. Wird von Javas Plattformunabhängigkeit Gebrauch gemacht, muss mit deutlichen Performance-Einbußen gerechnet werden. Insofern stellt Java für komplexe Anwendungen ein weniger gut geeignetes Entwicklungswerkzeug dar.

4.3.4 Vergleich und Bewertung

C++ war lange Zeit das Standardwerkzeug zur Entwicklung von Software-Komponenten. Aufgrund des wesentlich höheren Sicherheitsniveaus stellt Java in vielen Fällen das geeignetere Werkzeug dar. Sofern Performance und effiziente Ressourcennutzung nicht von übergeordneter Bedeutung sind, stellt Visual Basic ein gleichermaßen geeignetes Entwicklungssystem dar. Die wesentlichen Aspekte sind in Tabelle 4.3 dargestellt.

	C++	Visual Basic	Java
Hersteller	diverse	Microsoft	Sun Microsystems
Kapselung	ja	ja	ja
Polymorphie	ja	ja	ja
Implementierungsvererbung	mehrfach	nein	einfach
Separate Interfaces	nein	ja	ja
Typsicherheit	nein	ja	ja
Modulsicherheit	nein	nein	ja
Automatische Speicherverwaltung	nein	ja	ja

Tabelle 4.3: Komponentenorientierte Programmiersprachen

Kapitel 5
Architektur komponentenbasierter Software-Systeme

Eine überzogene Erwartungshaltung bezüglich der Vorteile der komponentenorientierten Software-Technik nährt die illusionäre Vorstellung, dass sich die Anwender zukünftig selbständig maßgeschneiderte Applikationen auf einfache und effiziente Weise aus völlig unabhängig voneinander entwickelten Software-Komponenten zusammenfügen können. Bei einer sachlichen Betrachtung wird jedoch offensichtlich, dass es extrem unwahrscheinlich ist, dass unabhängig entwickelte Software-Komponenten ohne eine ordnende Architektur überhaupt gemeinsam funktionieren können. Im folgenden Kapitel werden deshalb Strategien für den Entwurf von Architekturen komponentenbasierter Systeme vorgestellt und diskutiert.

5.1 Modellierungstechniken

Erfahrungsgemäß ist der Entwurf einer neuen Software-Architektur um Größenordnungen komplizierter als die Implementierung eines vorhandenen Designs. Die Entwicklung eines neuen Designs ist nach wie vor ein nicht-determinierter Prozess. Das heißt, er kann nicht durch Algorithmen beschrieben werden. Infolgedessen werden Erfolg oder Misserfolg im Wesentlichen von den Fähigkeiten und Erfahrungen der beteiligten Entwickler bestimmt. Wünschenswert ist es jedoch, Analyse und Design durch möglichst formalisierte Methoden zu unterstützen. Für die objektorientierte Software-Technik wurden zu diesem Zweck zahlreiche Methoden zur objektorientierten Modellierung vorgeschlagen (Rumbaugh et. al, 1991), (Booch, 1994), (Jacobson, 1993), die schließlich zur Entwicklung der Unified Modeling Language (Rumbaugh et al., 1997) und deren Erweiterungen führten.

5.1.1 Objektorientierte Modellierung

Die konsequente Anwendung der Konzepte der objektorientierten Software-Technik hat dazu geführt, dass mittlerweile auch komplexe Entwicklungsvorhaben beherrschbar sind. Insofern ergibt sich die Frage, inwieweit die dabei eingesetzten objektorientierten Modellierungstechniken für das Design von Systemen von Software-Komponenten angewendet werden können.

Obwohl eine umfassende Wiederverwendung von Software-Fragmenten einer der maßgeblichen Gründe für die Anwendung der objektorientierten Software-Technik war, wurde dieses Ziel nicht erreicht (Udell, 1994). Wesentliche Ursache dafür liegt im Paradigma der Vererbung und der damit einhergehenden White-Box-Abstraktion, die eine Wiederverwendung auf der Ebene des Quell-Codes anstatt durch binäre Bausteine erzwingen.

Konsequenterweise verzichten Komponentenmodelle entweder auf die Möglichkeit der Vererbung über die Grenzen von Software-Komponenten hinweg (z.B. COM) oder zumindest ist die Nutzung der gebotenen Möglichkeiten nicht ratsam (CORBA, Java). Somit kann die objektorientierte Modellierung aus technischen Gründen nur mit Einschränkungen zur Entwicklung von Software-Komponenten herangezogen werden.

Prinzipiell wäre es denkbar, ein mit Hilfe objektorientierter Modellierungsmethoden gefundenes Design durch eine Menge von Software-Komponenten nachzubilden, indem Vererbungsbeziehungen durch Mechanismen zur Komposition von Software-Komponenten ersetzt werden. Software-Komponenten mit der Granularität einzelner Klassen führen jedoch zu einem System mit extrem zahlreichen wechselseitigen Abhängigkeiten. Die Wiederverwendungsmöglichkeiten (vergleiche Kapitel 3.3.3) bleiben somit wie bei der objektorientierten Programmierung marginal.

Darüber hinaus beschränkt sich die objektorientierte Modellierung weitgehend auf technische Aspekte, wohingegen ökonomische Aspekte ignoriert werden. Diese haben jedoch für die erfolgreiche Entwicklung und Vermarktung von Software-Komponenten einen mindestens ebenbürtigen Stellenwert. Zusammenfassend muss festgestellt werden, dass eine einfache Übertragung der herkömmlichen objektorientierten Modellierungstechniken auf den Entwurf komponentenbasierter Systeme nicht empfehlenswert ist.

5.1.2 Alternative Modellierungstechniken

Als Alternative oder Ergänzung zu den gängigen objektorientierten Modellierungsmethoden wird gelegentlich eine Modellierung mit den Mitteln der Mathematik (insbesondere der Mengenlehre) vorgeschlagen (Laabs, 1998). Obwohl sich in der Tat viele Aspekte der Informatik auf mathematische Grundlagen zurückführen lassen, können damit ingenieurmäßige und ökonomische Gesichtspunkte wie Realisierbarkeit, Wiederverwendbarkeit, Qualität, Ressourcenbedarf, Entwicklungszeit und -kosten ebenfalls nicht erfasst werden. Außerdem konzentriert sich eine Modellierung auf Grundlage der Mengenlehre sehr stark auf die Identifizierung der Eigenschaften. Für eine komponentenorientierte Modellierung ist jedoch die Identifizierung der Methoden wesentlich wichtiger, da nur diese über Interfaces nach außen sichtbar gemacht werden.

Infolgedessen können mathematikbasierte Modellierungstechniken die Einschränkungen der objektorientierten Modellierung für den Entwurf komponentenbasierter Systeme nicht ausgleichen. Aus diesem Grund wird eine mathematikbasierte Modellierung als Alternative hier nicht weiter verfolgt.

5.1.3 Komponentenorientierte Modellierung

Da die herkömmlichen Modellierungstechniken versagen, erscheint es zweckmäßig, speziell auf die Erfordernisse von komponentenbasierten Systemen zugeschnittene Modellierungstechniken zu entwickeln. Die dazu benötigten Grundlagen werden jedoch nach dem gegenwärtigen Stand der Wissenschaft noch nicht hinreichend verstanden. Die Modellierung von komponentenbasierten Systemen stellt somit ein bislang weitgehend ungelöstes Problem dar.

Die wesentliche Schwierigkeit liegt darin, dass herkömmliche objektorientierte Modellierungstechniken für die vollständige Beschreibung von komplexen Systemen ausgelegt sind. Komponentenbasierte Systeme bilden jedoch niemals ein vollständiges System, da jederzeit weitere Software-Komponenten hinzugefügt werden können. Infolgedessen können derartige Systeme auch nicht global analysiert werden.

Während sich der Entwurf von Systemarchitekturen durch den komponentenorientierten Ansatz wesentlich komplizierter gestaltet, stellt das interne Design der Software-Komponenten geringere Anforderungen. Im Vergleich mit herkömmlichen Applikationen weisen Software-Komponenten eine deutlich geringere Komplexität auf. Software-Komponenten bestehen im Regelfall aus einer moderaten Anzahl kooperierender Objekte, die eine manchmal als Modul bezeichnete Einheit bilden. Da die interne Organisation nach außen nicht sichtbar ist, muss die Software-Komponente nicht zwingend objektorientiert implementiert sein, sondern kann beispielsweise auch prozedural strukturiert werden.

Zum jetzigen Zeitpunkt kann noch nicht abgeschätzt werden, ob und wann komponentenorientierte Modellierungsmethoden zur Verfügung stehen werden. Es erscheint jedoch wahrscheinlich, dass die Herausbildung eines Marktes für Software-Komponenten den Designprozess beeinflussen wird. Mit zunehmender Reife des Marktes für Software-Komponenten wird ein wachsender Anteil der Designentscheidungen durch die dazugekauften Software-Komponenten bestimmt werden.

Folglich wird zukünftig der erste Schritt in der Analyse der bereits verfügbaren Software-Komponenten liegen, in dem geklärt wird, welcher Teil der benötigten Funktionalität durch vorhandene Software-Komponenten abgedeckt werden soll. Anschließend kann ermittelt werden, welcher Teil der darüber hinaus benötigten Funktionalität durch selbst zu entwickelnde Software-Komponenten bereitgestellt werden soll. Zeitgleich mit der Partitionierung in Software-Komponenten müssen auch deren Interaktionsmöglichkeiten festgelegt werden. Die interne Organisation der Software-Komponenten ist bei diesem Prozess von untergeordnetem Interesse und kann als gekapseltes Detail völlig unabhängig vom Entwurf der Interfaces festgelegt werden.

5.2 Anforderungen an Komponenten-Software

Formalisierte Methoden zur Analyse und für den Entwurf komponentenbasierter Systeme stehen bislang nicht zur Verfügung. Trotzdem müssen vor dem Entwicklungsbeginn von Software-Komponenten folgenreiche Entscheidungen getroffen werden. Insbesondere müssen die folgenden Entwurfsparameter festgelegt werden:

- Granularität der Software-Komponenten
- Interaktionen zwischen den Software-Komponenten

Dabei beschreibt die Granularität die Größe der Software-Komponenten, während die Interaktionen im Wesentlichen durch die definierten Interfaces festgelegt werden. Das Treffen dieser frühen Designentscheidung ist ein subtiler Prozess, der den Erfolg der zu entwickelnden Software-Komponenten maßgeblich bestimmen wird.

Um die Eignung verschiedener Designszenarien zu beurteilen, muss abgeschätzt werden, wie gut die vorgegebenen Designziele erfüllt werden. Dabei werden die folgenden Designziele als wesentlich betrachtet:

- Rentabilität
- Wiederverwendbarkeit
- Robustheit
- Effizienz

Im Folgenden wird qualitativ abgeschätzt, welche Auswirkungen Änderungen der Granularität und der Interaktionen auf diese Designziele haben.

5.2.1 Rentabilität

Von den genannten Designzielen wird zwangsläufig der Rentabilität die größte Bedeutung beigemessen. Folglich lautet die zuerst zu beantwortende Frage, ob die Entwicklung einer Software-Komponente überhaupt gerechtfertigt ist. Eventuell sind vergleichbare Software-Komponenten bereits verfügbar, so dass die gewünschte Funktionalität ohne eigenes Entwicklungsrisiko eingekauft werden kann. Abbildung 5.1 illustriert, dass der dafür zu entrichtende Preis im Regelfall wesentlich unter den Kosten einer Eigenentwicklung liegt. Allerdings wird durch den Einkauf von Software-Komponenten der Gestaltungsspielraum des Entwicklungsprozesses eingeschränkt. Darüber hinaus ist die erworbene Funktionalität auch für Mitbewerber verfügbar. Insofern kann der Wunsch nach Flexibilität oder Wettbewerbsvorteilen dennoch die Entwicklung von eigenen Software-Komponenten rechtfertigen.

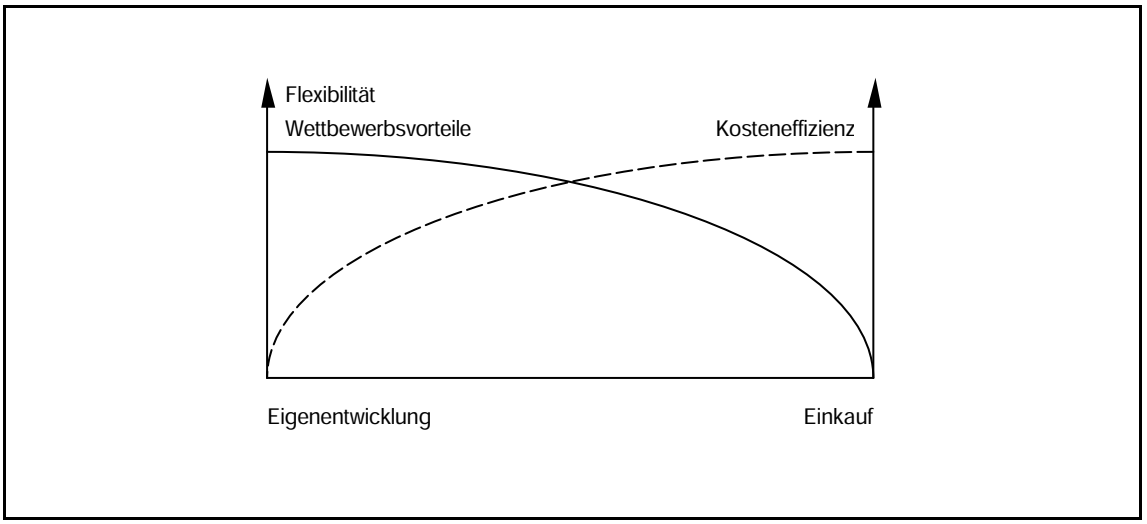


Abbildung 5.1: Kriterien für Eigenentwicklung oder Zukauf (Szyperski, 1998)

Wird die Entwicklung einer eigenen Software-Komponente beschlossen, sollen die damit verbundenen Investitionen möglichst rentabel sein. Der zur Realisierung erforderliche Aufwand ist im Vergleich zu herkömmlichen Implementierungsstrategien wesentlich höher, da die Lösung eines generalisierten Problems stets anspruchsvoller ist als die Lösung eines konkreten Problems.

Darüber hinaus steigt mit der Zahl der Anwendungsmöglichkeiten der Aufwand für die Erstellung der Dokumentation und zum Testen der Software-Komponenten. Dieser erhöhte Aufwand kann zum einen direkt über die mit dem Verkauf der Software-Komponenten erwirtschafteten Erträge gedeckt werden. Zum anderen kann der Mehraufwand bei einer firmeninternen Nutzung von Software-Komponenten auch durch indirekte Vorteile wie kürzere Entwicklungszeiten, bessere Beherrschbarkeit, Wartbarkeit, Konfigurierbarkeit und Flexibilität gerechtfertigt sein.

Es erscheint wahrscheinlich, dass der Handel mit Software-Komponenten dazu führen wird, dass Software-Hersteller sich auf die Entwicklung von Software-Komponenten innerhalb ihrer Kernkompetenzen konzentrieren und die darüber hinaus benötigte Funktionalität hinzukaufen.

5.2.2 Wiederverwendbarkeit

Die maßgebliche Motivation für die Entwicklung von Software-Komponenten stellt deren Wiederverwendbarkeit dar, da sich dann Investitionen in Software-Komponenten über mehrere Anwendungen hinweg amortisieren können.

Folglich werden Software-Komponenten im Hinblick auf eine Komposition durch Dritte entwickelt. Damit ausreichende Wiederverwendungsmöglichkeiten von Software-Komponenten erreicht werden, müssen diese sorgfältig generalisiert und deren Kontextabhängigkeiten gering gehalten werden. Andererseits kann eine Software-Komponente, die extrem generalisiert und völlig unabhängig von einem bestimmten Kontext ist, überhaupt nicht sinnvoll eingesetzt werden (Murer, 1997). Eine übertriebene Generalisierung muss also vermieden werden. Übergeneralisierung kann sich beispielsweise in einem Abstraktionsgrad äußern, der von den potenziellen Nutzern der Software-Komponente nicht mehr interpretiert werden kann.

Neben den Wiederverwendungsmöglichkeiten der Software-Komponenten selbst spielt auch die Wiederverwendbarkeit von Implementierungsfragmenten innerhalb von Software-Komponenten eine Rolle. Im einfachsten Fall wird eine Software-Komponente die gesamte von ihr benötigte Funktionalität selbst bereitstellen. Alternativ dazu kann der Entwickler zur Vermeidung von Redundanzen alles – mit Ausnahme der primären Funktionalität – extrahieren, generalisieren und in eigenständige Software-Komponenten verlagern, so dass eine Menge von generischen Software-Komponenten entsteht. Eine derartige Vorgehensweise folgt im Wesentlichen den propagierten Methoden von objektorientierter Analyse und Design.

Dem Vorteil der maximalen Wiederverwendung von Code stehen jedoch gravierende Nachteile gegenüber. Mit der zunehmenden Anzahl der beteiligten Software-Komponenten erhöht sich die Anzahl der wechselseitigen Abhängigkeiten drastisch. Infolge der umfangreichen Kontextabhängigkeiten wird das Anwendungspotential der Software-Komponenten reduziert. Entwickler müssen somit die Vorteile einer höheren Wiederverwendung von Code gegen das durch zunehmende Kontextabhängigkeiten verkleinerte Marktpotential abwägen.

5.2.3 Robustheit

Ein System von Software-Komponenten ist nur so zuverlässig wie seine schwächste Software-Komponente. Infolgedessen kommt der Robustheit der eingesetzten Software-Komponenten große Bedeutung zu. Wie bereits zuvor erläutert, reduziert sich mit zunehmender Zahl von Kontextabhängigkeiten die Wahrscheinlichkeit, dass alle sich wechselseitig benötigenden Software-Komponenten vorhanden sind. Infolge der unabhängig voneinander erfolgenden Weiterentwicklung der Software-Komponenten müssen diese die Zusammenarbeit mit unterschiedlichen Versionen unterstützen. Somit werden die durch Kontextabhängigkeiten hervorgerufenen Probleme nochmals vervielfacht. Die Zuverlässigkeit eines Systems von Software-Komponenten sinkt mit zunehmender Komplexität.

5.2.4 Effizienz

Für den wirtschaftlichen Erfolg einer Software-Komponente spielt die Effizienz, das heißt ein niedriger Ressourcenverbrauch an Rechenleistung und Arbeitsspeicher, eine entscheidende Rolle. Die Effizienz wird maßgeblich durch die Granularität der Software-Komponenten bestimmt. Die Granularität von Software-Komponenten sollte deshalb so festgelegt werden, dass die Mehrzahl der Interaktionen innerhalb der Grenzen einer Software-Komponente stattfindet. Die Zerlegung einer Anwendung in kleinste Software-Komponenten erlaubt zwar die maximale Wiederverwendung von Code, kollidiert aber mit dem Ziel Effizienz. Die Kommunikation über Komponentengrenzen und insbesondere Prozessgrenzen ist um Größenordnungen aufwändiger, so dass sich die Effizienz reduziert. Aus diesem Grund wurde beispielsweise die ursprüngliche Micro-Kernel-Architecture von Windows NT 3.5 bei der Weiterentwicklung zu Windows NT 4.0 wieder abgeschwächt.

5.2.5 Fazit

In der vorangegangenen Analyse wurde gezeigt, dass eine Verringerung der Granularität von Software-Komponenten eine Reduzierung des Anwendungspotentials bedeutet, die mit einer Verringerung von Robustheit und Effizienz einhergeht. Der einzige Beweggrund, der für eine feine Granularität spricht, ist die Vermeidung von redundanten Implementierungen (Abbildung 5.2).

Somit erfordern nahezu alle Aspekte, die zur Festlegung der Granularität relevant sind, verhältnismäßig grobe Strukturen. Eine Granularität in der Größenordnung von einzelnen Klassen oder Prozeduren verbietet sich im Regelfall von selbst, da offensichtlich durch das Anstreben von maximaler Wiederverwendbarkeit die Einsatzmöglichkeiten von Software-Komponenten minimiert werden.

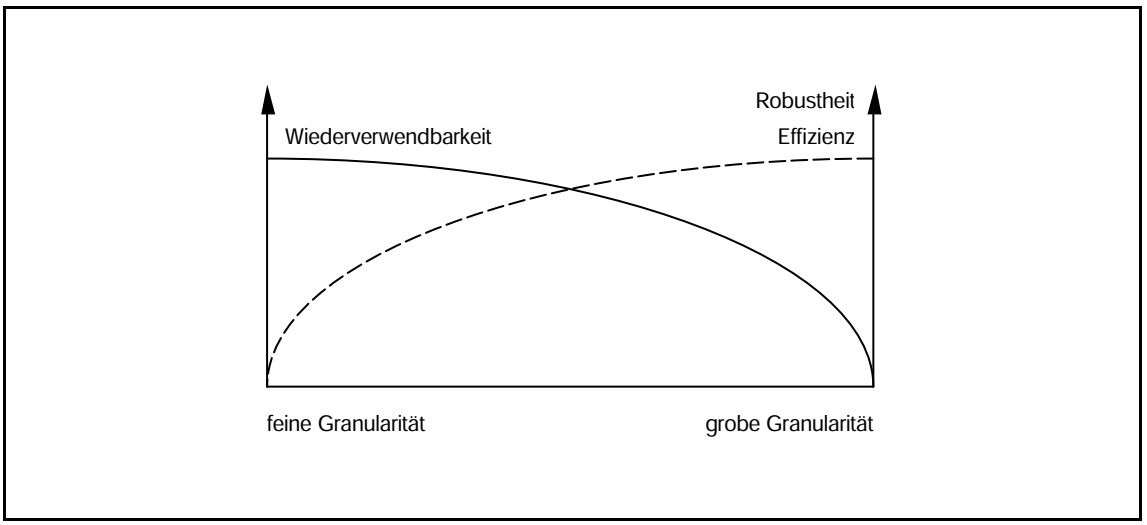


Abbildung 5.2: Granularität versus Wiederverwendbarkeit, Robustheit und Effizienz (Szyperski, 1998)

5.3 Designstrategien

Fehler beim Design einer Software-Architektur können weitreichende Konsequenzen haben, da zu deren Behebung die vorher definierten Interfaces geändert werden müssen, was seinerseits eine Modifikation der bestehenden Implementierungen erforderlich macht. Die Wiederverwendung eines Designs ist folglich wichtiger als die Wiederverwendung von Interfaces und wesentlich wichtiger als die Wiederverwendung von Programm-Code (Pfister, 1997).

In Ermangelung von Alternativen erscheint es angebracht, auf die traditionellen Hilfsmittel zur Wiederverwendung von Designs zurückzugreifen und diese soweit wie möglich an die Erfordernisse von Komponenten-Software anzupassen. Zu den bewährten Hilfsmitteln gehören Design Patterns und Class Frameworks. Übertragen auf das Gebiet der Komponenten-Software spielen die folgenden drei Designstrategien eine wesentliche Rolle:

- Patterns
- Component Frameworks
- Component Systems

5.3.1 Patterns

In (Gamma et al., 1995) ist der Versuch beschrieben, die kleinsten wiederkehrenden Architekturmuster in objektorientierten Systemen systematisch zu identifizieren und zu katalogisieren. Derartige Muster werden als Design Patterns bezeichnet. Design Patterns stellen Mikroarchitekturen dar, die auf einer abstrakten Ebene beschreiben, welche Interaktionen von Objekten zur Lösung eines bestimmten Problemtyps erforderlich sind. Dieses Schema kann auf ähnliche Problemklassen übertragen und an deren konkrete Erfordernisse angepasst werden. Dadurch wird eine Wiederverwendung des Lösungsansatzes ermöglicht.

Neben den Wiederverwendungsmöglichkeiten bieten Design Patterns einen weiteren Vorteil. Da jedes katalogisierte Design Pattern einen Namen erhält, können die komplexen Konzepte durch einen prägnanten Begriff identifiziert werden. Eine effiziente Kommunikation der beteiligten Entwickler wird dadurch wesentlich erleichtert.

Für jedes Pattern werden die folgenden vier Angaben gemacht:

- Name des Patterns
- Problem, das durch das Pattern gelöst wird
  Voraussetzungen, damit das Pattern angewendet werden kann
- Beschreibung, wie das Pattern das Problem löst
  Beteiligte Elemente, deren Verantwortlichkeiten und Beziehungen
  Abstrakte Beschreibung der allgemeinen Lösung, jedoch keine Details der Implementierung
- Konsequenzen der Anwendung des Patterns
  Abschätzung von Kosten und Nutzen
  Diskussion von Sprach- und Implementierungsfragen
  Auswirkungen in Bezug auf Flexibilität, Erweiterbarkeit und Portabilität

Ein Beispiel mit hohem Bekanntheitsgrad stellt das Observer Pattern dar, das gemeinsam mit den Pattern Composite and Strategy zur Implementierung des Model-View-Controller-Paradigmas benutzt werden kann. Diesbezügliche Details werden in (Gamma et al., 1995) diskutiert.

Im Vergleich zu den anschließend betrachteten Frameworks sind Design Patterns abstrakter und weniger spezialisiert. Während Frameworks zumeist eine Implementierung beinhalten, stellen Patterns lediglich ein Lösungsmuster dar. Patterns stellen wesentlich kleinere architektonische Elemente dar als Frameworks, wohingegen Frameworks gewöhnlich mehrere Patterns nutzen. Frameworks werden stets für bestimmte Fachgebiete entwickelt, wohingegen Design Patterns weniger spezialisiert sind und im Regelfall nicht an ein bestimmtes Anwendungsgebiet gebunden sind.

5.3.2 Frameworks

Bisher hat der Schwerpunkt der komponentenorientierten Programmierung auf der Entwicklung einzelner Software-Komponenten gelegen. Die Möglichkeit Software-Komponenten unabhängig voneinander zu entwickeln, zu vertreiben und später zu maßgeschneiderten Applikationen zusammenzufügen, stellt das langfristige, aber bisher nicht verwirklichte Ziel der komponentenorientierten Software-Technik dar. Die dafür benötigte Basistechnologie bilden unabhängig erweiterbare Systeme. Nach (Szyperski, 1996) gilt ein System als unabhängig erweiterbar, wenn es erweitert werden kann und wenn es die Kombination von unabhängig voneinander entwickelten Elementen erlaubt. Das Konzept der unabhängigen Erweiterbarkeit ist keineswegs neu: Jedes Betriebssystem erlaubt zur Laufzeit die Installation von unabhängig voneinander entwickelten Anwendungen, ermöglicht diesen miteinander zu kommunizieren und reguliert die gemeinsame Nutzung von Ressourcen.

Eine Interoperabilität von unabhängig voneinander entwickelten Software-Komponenten ist jedoch nur möglich, wenn diese konform zu bestimmten Standards oder Konventionen sind. Offensichtlich sind derartige Richtlinien von dem jeweiligen Anwendungszweck abhängig. In (Weck, 1997b) wird vorgeschlagen, dass die Einhaltung dieser domänenspezifischen Designprinzipen durch sogenannte Component Frameworks sichergestellt wird.

5.3.2.1 Class Frameworks

Der Begriff Framework wurde ursprünglich im Zusammenhang der objektorientierten Programmierung als eine Anzahl von kooperierenden Klassen definiert, die ein wiederverwendbares Design für ein konkretes Anwendungsgebiet darstellen (Deutsch, 1989).

Die generalisierte Lösung des Class Frameworks für ein bestimmtes Problem wird durch die Ableitung anwendungsspezifischer Subklassen an die jeweiligen Erfordernisse angepasst. Dieser Prozess wird als Spezialisierung bezeichnet. Class Frameworks enthalten im Normalfall konkrete Klassen, deren Abstraktionen von den Entwicklern direkt wiederverwendet werden können. Infolgedessen stellen Class Frameworks White-Box-Architekturen dar, die auf eine Veröffentlichung des Quell-Codes angewiesen sind (vergleiche Kapitel 3.3.1). Der Schwerpunkt der Wiederverwendung liegt dabei auf der Implementierung.

Neben der Implementierung sind in jedem Class Framework auch bestimmte architektonische Prinzipien fixiert. Durch die vorgegebene Partitionierung in Klassen und den Interaktionen der Objekte sind bestimmte Abstraktionen inhärent, so dass die damit erzeugten Applikationen zwangsläufig eine ähnliche Struktur aufweisen. Insofern ermöglichen Class Frameworks auch eine Wiederverwendung des Designs.

Class Frameworks stellen einen erfolgreichen Ansatz dar, der aber auch mit gravierenden Einschränkungen verbunden ist. So ist eine Spezialisierung ausschließlich zum Zeitpunkt der Implementierung möglich. Danach erfolgt eine Verschmelzung des Class Frameworks und des selbst geschriebenen Codes. Eine unabhängige Weiterentwicklung von Class Framework und Implementierung ist wegen des Fragile-Base-Class-Problems (vergleiche Kapitel 3.3.2) nur sehr eingeschränkt möglich. Noch schwieriger gestaltet sich die Migration zwischen zwei Class Frameworks mit vergleichbarer Funktionalität.

5.3.2.2 Component Frameworks

Obwohl Class und Component Frameworks vergleichbare Ziele anstreben und scheinbar den gleichen Konstruktionsprinzipien folgen, sind beide Ansätze sehr verschieden. Traditionelle Class Frameworks sind auf Klassen und Vererbung ausgerichtet, benötigen dafür Informationen über die Implementierung der Superklassen und stellen somit White-Box-Frameworks dar. Allerdings ist es ebenso möglich, Frameworks zu entwickeln, die auf der Komposition von Software-Komponenten basieren, dafür lediglich Kenntnis der Interfaces sowie deren Spezifikation benötigen und somit Black-Box-Architekturen darstellen. Derartige Frameworks werden als Component Frameworks bezeichnet.

Das wesentliche Merkmal eines Component Frameworks ist es, dass es zur Laufzeit die dynamische Integration von Software-Komponenten ermöglicht. Dabei bleiben die Implementierungen des Component Frameworks und der Software-Komponenten strikt voneinander getrennt. Da im Unterschied zu Class Frameworks keine Verschmelzung stattfindet, können Software-Komponenten auch wieder deinstalliert oder durch andere ersetzt werden. Ein derartiges Konzept wird als Plug-in-Architecture bezeichnet.

Die Architektur des Systems von Software-Komponenten wird durch die Entwicklung eines Component Frameworks weitgehend vordefiniert. Durch die Fixierung bestimmter Entwurfsüberlegungen wird der Entwicklungsprozess vereinfacht, was zu einer Zeitersparnis führen kann. Gleichzeitig wird das Entstehen zufälliger Strukturen mit redundanten oder inkonsistenten Lösungen vermieden.

Ein Component Framework basiert wie jedes unabhängig erweiterbare System auf den Prinzipien der Koexistenz. Aufgabe des Component Frameworks ist es deshalb, als übergeordnete Instanz die Interaktionen zwischen den Software-Komponenten zu vermitteln und zu regulieren. Ohne eine derartige Infrastruktur wäre eine Interoperabilität der unabhängig voneinander entwickelten Software-Komponenten ausgeschlossen.

Technisch gesehen stellt ein Component Framework eine Menge von Interfaces dar, durch die die Interaktionsmöglichkeiten der Software-Komponenten geregelt werden. Es wäre zwar theoretisch vorstellbar, dass ein Component Framework lediglich eine abstrakte Spezifikation von zu befolgenden Regeln darstellt. Praktisch hat die bloße Spezifikation von Regeln keinen Nutzen, da nicht sichergestellt ist, dass die Software-Komponenten diese auch befolgen. Stattdessen sollte die Einhaltung der Konventionen durch das Component Framework erzwungen werden (Weck, 1997b). Zu diesem Zweck werden die einzelnen Software-Komponenten verpflichtet, bestimmte Vorgänge wie zum Beispiel das Versenden von Nachrichten über Mechanismen ablaufen zu lassen, die vom Component Framework kontrolliert werden.

Die Entwicklung eines Component Frameworks nimmt wesentliche Designentscheidungen bezüglich der späteren Software-Komponenten vorweg und stellt somit ein Meta-Design dar. Erst die Entwicklung individueller Software-Komponenten ermöglicht Rückschlüsse auf die Zweckmäßigkeit der getroffenen Designentscheidungen. Die Entwicklung eines Component Frameworks kann somit nur ein iterativer Prozess sein.

Bislang hat noch kein Component Framework nennenswerte Verbreitung gefunden. Die vielversprechende Entwicklung des OpenDoc-Frameworks (vergleiche Kapitel 4.2.2) für Verbunddokumente wurde vor dessen Fertigstellung eingestellt. Zu den wenigen derzeit erhältlichen Component Frameworks gehört das von Oberon Microsystems entwickelte BlackBox Component Framework (Pfister, 1997). Dessen Verbreitung ist jedoch wie die der zugehörigen Entwicklungsumgebung BlackBox Component Builder marginal. Im Gegensatz zu anders lautenden Aussagen stellt OLE kein Component Framework dar. Zwar werden zahlreiche Konventionen in der OLE-Spezifikation aufgeführt, deren Einhaltung aber nicht einheitlich erzwungen. Vielmehr unterstützt jeder OLE-Container eine Teilmenge der OLE-Interfaces und stellt somit ein individuelles Component Framework dar. Möglich erscheint jedoch, dass sich Microsofts Internet Explorer als Standard für einen generischen Container für ActiveX-Controls etabliert.

5.3.2.3 Dimensionen unabhängiger Erweiterbarkeit

Um die Interoperabilität zwischen den unabhängigen Erweiterungen zu gewährleisten, darf das Component Framework nur in der ursprünglich vorgesehenen Art und Weise erweitert werden. Die dabei zulässigen Erweiterungsrichtungen werden als Dimensionen der Erweiterbarkeit bezeichnet. Offensichtlich ist es nicht realistisch, zum Zeitpunkt des Designs des Component Frameworks alle möglichen Erweiterungen detailliert zu beschreiben. Die Herausforderung besteht darin, die benötigten Dimensionen der Erweiterbarkeit vorab zu identifizieren.

Die möglichen Dimensionen unabhängiger Erweiterbarkeit müssen nicht notwendigerweise orthogonal sein. Dieselben Effekte können unter Umständen durch Erweiterung in verschiedenen Dimensionen erreicht werden. Dies ist im Regelfall nicht erstrebenswert, da dadurch für die Entwickler ein unerwünschter Gestaltungsspielraum geschaffen wird. Eine perfekte Orthogonalität kann jedoch nur selten erreicht werden, so dass gewisse Redundanzen in Kauf genommen werden müssen. Ein vollständiger Verzicht auf redundante Erweiterungen kann eine starke Einschränkung darstellen. Die Folge wäre ein orthogonales, aber unvollständiges System.

Enthält eine Konfiguration eines Component Frameworks mehrere Erweiterungen entlang derselben Dimension, so wird dies als parallele Erweiterung bezeichnet (Weck, 1997b). Da parallele Erweiterungen ähnliche Aufgaben erfüllen, besteht die Gefahr, dass sie dieselben Ressourcen benötigen. Das Component Framework ist folglich vordergründig dafür verantwortlich, dass eine Koexistenz der parallelen Erweiterungen sichergestellt wird.

Enthält eine Konfiguration eines Component Frameworks Erweiterungen entlang verschiedener Dimensionen, so wird dies als orthogonale Erweiterung bezeichnet (Weck, 1997b). Die Schwierigkeiten bezüglich orthogonaler Erweiterungen liegen in der Bereitstellung von Interfaces, die eine Interaktion der Erweiterungen ermöglichen.

Erlaubt ein Component Framework die Erweiterung in einer obligatorischen Dimension durch genau eine oder bei optionalen Dimensionen durch höchstens eine Software-Komponente, so wird dies als Singleton-Konfiguration bezeichnet (Szyperski, 1998). Eine Singleton-Konfiguration ist beispielsweise für Dimensionen der Erweiterung zweckmäßig, die als kritisch für die Sicherheit des Systems betrachtet werden.

Das Konzept der Dimensionen unabhängiger Erweiterbarkeit soll im Folgenden am Beispiel eines Component Frameworks für Verbunddokumente erläutert werden. Ein derartiges Framework kann um zusätzliche Typen von Containern und Controls ergänzt werden, solange diese den Konventionen des Frameworks folgen. Container und Controls haben dabei völlig verschiedene Aufgaben und stellen deshalb Erweiterungen in zwei orthogonalen Dimensionen dar. Alle Controls können in jeden beliebigen Container eingefügt werden. Aus der Sicht der Controls sind die verschiedenen Containertypen gleichwertig und stellen somit parallele Erweiterungen dar. Umgekehrt spielt es aus der Sicht eines Containers keine Rolle, welche Controls er beinhaltet, so dass die verschiedenen Typen von Controls ebenfalls parallele Erweiterungen darstellen.

5.3.3 Component Systems

Die durch ein Component Framework definierten Regeln sind abhängig vom betrachteten Fachgebiet, für das die Software-Komponenten Lösungen implementieren sollen. Daraus folgt, dass es entsprechend der Vielzahl an Fachgebieten auch eine Vielzahl von Component Frameworks geben wird. Komplexere Systeme werden zur Erfüllung ihrer Aufgaben mehrere Component Frameworks benötigen.

Die bisher anvisierten Component Frameworks schaffen jedoch nur ein isoliertes System von Software-Komponenten. Es muss jedoch auch möglich sein, Interoperabilität zwischen mehreren Component Frameworks zu erreichen. Derartige Systeme werden in (Szyperski, 1998) als Component Systems bezeichnet. Gegenwärtig liegen keinerlei Erfahrungen mit Component Systems vor. Im Folgenden sollen die in der Literatur diskutierten Ansätze gegenübergestellt werden.

5.3.3.1 Tiered Component Frameworks

Die Gliederung komplexer Systeme in hierarchische Strukturen stellt eine bewährte Strategie des Software-Ingenieurwesens dar. In (Szyperski, 1998) wird deshalb das Konzept von Tiered Component Frameworks vorgeschlagen. Dabei bilden mehrere Component Frameworks ein hierarchisches System. Wie in Abbildung 5.3 dargestellt werden Component Frameworks dabei ihrerseits als Software-Komponenten aufgefasst, die Bestandteil eines Component Frameworks höherer Ordnung sind.

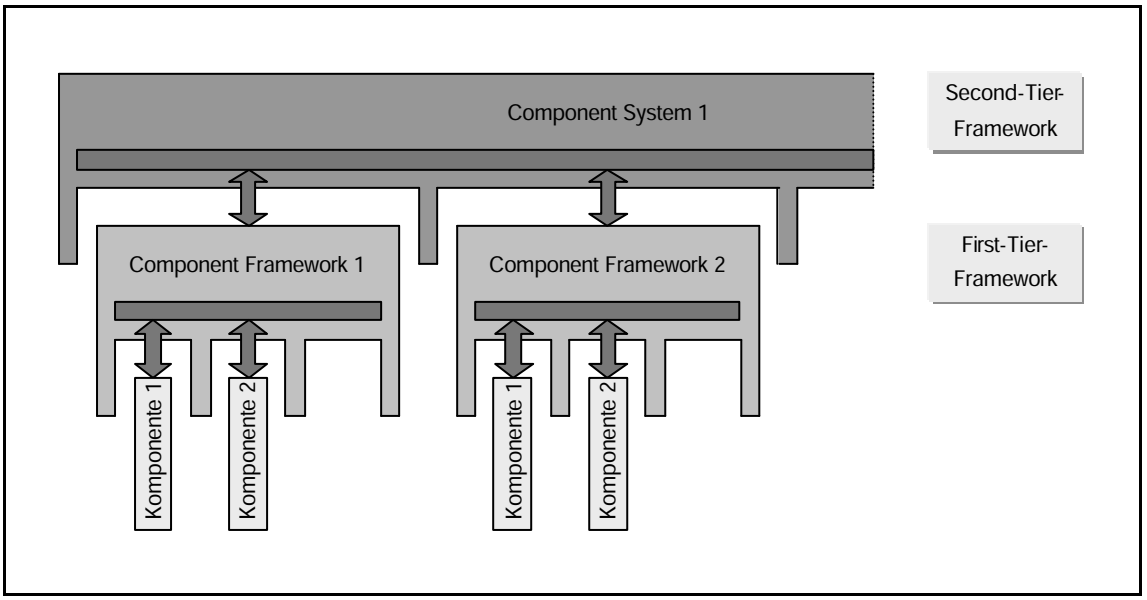


Abbildung 5.3: Software-Komponenten, Component Frameworks, Component Systems

Obwohl Component Frameworks höherer Ordnung langfristig unbedingt notwendig erscheinen, sind sie bislang nicht existent. Die Entwicklung eines allgemeinen Ansatzes für ein Component System mit beliebig vielen Hierarchien ist deshalb zur Zeit nicht realistisch. Deshalb wird in (Szyperski, Vernik, 1998) vorgeschlagen, sich zunächst auf eine 2-Tier-Architektur zu beschränken.

Die erste Ebene enthält die domänenspezifischen Software-Komponenten. Die zweite Ebene soll Interoperabilität und Integration zwischen den verschiedenen Fachbereichen ermöglichen. Folglich werden im Regelfall 2-Tier-Component Frameworks als ausreichend erachtet.

5.3.3.2 Federated Component Frameworks

In (Graw, Mester, 1998) wird ebenfalls ein Component System vorgeschlagen. Es besteht jedoch aus mehreren gleichberechtigten Component Frameworks, die als Federated Component Frameworks bezeichnet werden. Im Unterschied zum Konzept des Tiered Component Frameworks sollen die Component Frameworks mehr Autonomie bezüglich der Kooperationsmöglichkeiten, des Interface Designs und der Weiterentwicklungsmöglichkeiten erhalten. Um Konflikte aufzulösen und so dennoch Interaktionsmöglichkeiten zwischen den weitgehend unabhängigen Component Frameworks zu erreichen, sollen sogenannte Trader eingesetzt werden. Damit sind Federated Component Frameworks besonders für Spezialaufgaben geeignet, die eine hohe dynamische Flexibilität erfordern.

Kapitel 6
Gestaltung ingenieurgemäßer Software

Gegenstand dieses Kapitels sind verschiedene Konzepte, die aus der Sicht der Tragwerksplaner zu ingenieurgemäß gestalteter Tragwerksplanungs-Software beitragen können. Zu diesem Zweck werden zunächst die Arbeitsabläufe in der Tragwerksplanung analysiert und verschiedene Varianten der Modellierung von Tragwerken gegenübergestellt. Danach werden die Merkmale von ergonomisch gestalteten Nutzeroberflächen benannt. Außerdem werden Fragen der Integration von Tragwerksplanungs-Software diskutiert, insbesondere im Hinblick auf die Einbindung von CAD-Software und die Realisierung des Lastabtrages.

6.1 Arbeitsweise des Ingenieurs

Ein umfassendes Verständnis der Arbeitsweise des Tragwerksplaners ist die wichtigste Voraussetzung für die ingenieurgemäße Gestaltung von Tragwerksplanungs-Software. Im Folgenden wird deshalb der Arbeitsablauf der Tragwerksplanung diskutiert. Besonderer Stellenwert wird auch Fragen der Modellbildung eingeräumt.

6.1.1 Ablauf der Tragwerksplanung

Der Prozess der Tragwerksplanung gliedert sich grundsätzlich in fünf Phasen, die gegebenenfalls mehrfach durchlaufen werden.

1. Entwurf des Tragwerkes

   Während dieser Phase werden sogenannte Positionspläne erstellt, die als Grundlage für die nachfolgenden Arbeitsschritte dienen. Als Arbeitswerkzeug wird im Regelfall CAD-Software eingesetzt. Der Tragwerksplaner greift dabei auf Unterlagen zurück, die ihm vom Architekten oder Objektplaner zur Verfügung gestellt werden.

2. Idealisierung von Tragwerk und Einwirkungen

   Die Informationen aus der Entwurfsphase werden durch den Tragwerksplaner in statische Systeme umgesetzt. Dabei wird das Tragwerk in als Positionen bezeichnete Teiltragwerke zerlegt. Während dieses Vorgangs werden die Geometrie vereinfacht (z.B. durch Weglassen von Aussparungen oder Details), die Materialeigenschaften, die Auflagerbedingungen sowie die Bauteildimensionen vorläufig festgelegt. Um den Rechenaufwand zu verringern, wird häufig eine Dimensionsreduktion vorgenommen, das heißt, 3D-Bauteile werden durch äquivalente 2D-Tragwerke (Platten, Scheiben) oder 1D-Tragwerke (Balken, Stützen) ersetzt. Außerdem müssen Annahmen bezüglich der Einwirkungen getroffen werden, für die das Tragverhalten untersucht werden soll. Der Prozess der Idealisierung wird bislang nicht nennenswert durch Software unterstützt.

3. Ermittlung der Schnittgrößen

   Mit den als Ergebnis der Idealisierungsphase vorliegenden Informationen können Schnittgrößen sowie Spannungen und Verformungen ermittelt werden. Die Ermittlung erfolgt in den meisten Fällen mit Software, in einfachen Fällen jedoch auch durch Handrechnung oder tabellierte Lösungen.

4. Bemessung/Nachweis

   Es wird überprüft, ob die vorläufig festgelegten Bauteildimensionen und Materialeigenschaften zweckmäßig sind. Gegebenenfalls müssen die geometrischen oder materiellen Eigenschaften modifiziert werden. Im Anschluss daran müssen die Phasen Idealisierung und Schnittgrößenermittlung erneut durchlaufen werden. Einen Sonderfall stellt der Massivbau dar, wo die Tragfähigkeit im Wesentlichen durch die vorhandenen Bewehrungsmengen bestimmt wird, so dass die Bauteildimensionen im Regelfall beibehalten werden können. Im Anschluss an die Nachweise der Tragfähigkeit wird die Gebrauchstauglichkeit des Teiltragwerkes nachgewiesen. Die Bemessung beziehungsweise Nachweisführung wird teilweise durch Software unterstützt.

5. Konstruktive Durchbildung

   Im letzten Schritt wird eine detaillierte Konstruktion entworfen, die den Erfordernissen der Bauausführung entspricht. Außerdem werden Effekte, die bei der Modellbildung oder der Berechnung nicht erfasst wurden, durch konstruktive Maßnahmen berücksichtigt. Der Prozess der konstruktiven Durchbildung wird nur zu einem geringen Teil durch Software unterstützt.

Das Ergebnis dieser 5 Phasen wird vom Tragwerksplaner im Tragwerksbericht und in Plänen dokumentiert und von einem unabhängigen Prüfingenieur begutachtet.

6.1.2 Modellierung von Tragwerken

Prinzipiell existieren zwei gegensätzliche Ansätze zur Modellbildung von Tragwerken (Abbildung 6.1):

- Modellierung als dimensionsreduzierte Teiltragwerke
- Modellierung als dreidimensionales Kontinuum

Die herkömmliche Vorgehensweise besteht darin, dass der Ingenieur das Tragwerk in weitgehend unabhängig voneinander untersuchte Positionen zerlegt. Die Positionen entsprechen in der Regel Bauteilen. Bei der Zerlegung wird versucht, die wesentlichen Phänomene des Tragverhaltens zu berücksichtigen. Dennoch stellt eine derartige Modellbildung eine weitreichende Idealisierung des realen physikalischen Verhaltens dar. Neben den durch die Dimensionsreduzierung bewirkten Vereinfachungen werden auch die Interaktionen (z.B. infolge Verformung) zwischen den Positionen weitgehend vernachlässigt. Die Komplexität der Tragwerksanalyse wird dadurch wesentlich verringert. Allerdings muss infolge der gedanklichen Trennungen der Lastabtrag zwischen den einzelnen Positionen explizit festgelegt werden. Außerdem müssen Effekte, die mit der gewählten Modellbildung nicht erfasst werden können, gesondert berücksichtigt werden.

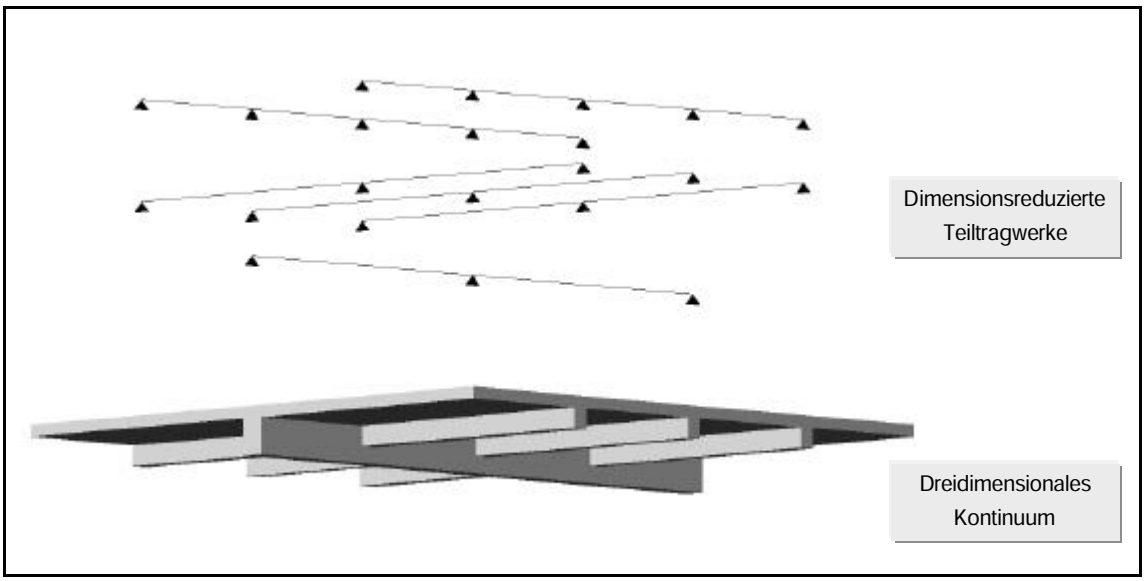


Abbildung 6.1: Möglichkeiten der Modellbildung von Tragwerken

Infolge der zunehmenden Leistungsfähigkeit von Hardware und Software können immer komplexere Tragwerke analysiert werden. Prinzipiell ist es heute möglich, komplette Bauwerke als dreidimensionales Kontinuum zu modellieren. Ein derartiges Modell kann beispielsweise mit Hilfe der Finite-Element-Methode analysiert werden. Da sämtliche Interaktionen zwischen den einzelnen Tragwerksbestandteilen erfasst werden können, ermöglicht diese Vorgehensweise theoretisch eine sehr hohe Genauigkeit der Ergebnisse. Praktisch ist die derzeit verfügbare Rechenleistung für die Analyse von 3D-Modellen selbst bei Tragwerken mit durchschnittlicher Komplexität nicht ausreichend. Um dennoch zu zumutbaren Rechenzeiten zu kommen, wird häufig mit einer zu groben Diskretisierung gearbeitet, die das reale Tragverhalten nicht ausreichend gut annähern kann. Zu den Diskretisierungsfehlern kommen insbesondere bei komplexen Tragwerken und dementsprechend großen Gleichungssystemen noch numerische Ungenauigkeiten. Da jedoch auch bei Anwendung eines 3D-Modells weitreichende Vereinfachungen und Annahmen erforderlich sind (z.B. bezüglich der Einwirkungen, des Materialverhaltens, des Baugrundes), ist die scheinbar mögliche Steigerung der Genauigkeit über ein bestimmtes Maß hinaus ohnehin nicht sinnvoll. Vorteilhaft ist, dass die Schnittgrößen bei Änderungen in einem Schritt für das gesamte Tragwerk neu ermittelt werden können. Anschließend müssen jedoch die relevanten Änderungen der Schnittgrößen lokalisiert werden. Damit muss die Bemessung und die konstruktive Durchbildung wiederholt werden. Das bedeutet, dass selbst geringfügige lokale Änderungen weitreichende Auswirkungen auf die gesamte Tragstruktur haben können. Überhaupt ist eine ingenieurmäßige Abschätzung des Tragverhaltens infolge der hohen Komplexität kaum noch möglich. Infolgedessen besteht die Gefahr, dass der Tragwerksplaner den Überblick verliert und Fehler übersieht, die bei einfacheren Modellen offensichtlich wären.

Die positionsweise Analyse von Tragwerken entspricht eher der Arbeits- und Denkweise der Tragwerksplaner. Folglich wird die Mehrzahl der Tragwerke weiterhin auf herkömmliche Art und Weise analysiert. Dies wird sich auch vorerst nicht ändern.

6.2 Ergonomische Nutzeroberflächen

Aus der Sicht des Anwenders stellen ergonomische Nutzeroberflächen ein wesentliches Qualitätsmerkmal von Software dar. Die Gebrauchstauglichkeit von Software wird im Wesentlichen durch drei Kriterien bestimmt: Der Anwender erwartet, dass er mit Hilfe der eingesetzten Software seine Aufgaben nicht nur lösen kann (Effektivität), sondern dabei nicht mehr Zeit als nötig aufwendet (Effizienz) und dennoch nicht sklavisch den Vorgaben der Technik folgen muss (Zufriedenheit).

An ergonomische Nutzeroberflächen werden deshalb die folgenden Anforderungen gestellt:

- die Benutzerführung muss dem Arbeitsablauf entsprechen
- die Informationen müssen konsistent dargestellt werden
- die Informationen müssen übersichtlich dargestellt werden

Von den genannten Anforderungen ist insbesondere eine dem Arbeitsablauf angepasste Benutzerführung wichtig, da diesbezügliche Mängel nach einer von (System Concepts, 1997) durchgeführten Untersuchung für ungefähr 60% aller Nutzungsprobleme verantwortlich sind. Lediglich 25% der Nutzungsprobleme sind auf eine inkonsistente Gestaltung und nur 15% auf ungünstige räumliche Platzierung der Dialogelemente zurückzuführen (Abbildung 6.2).

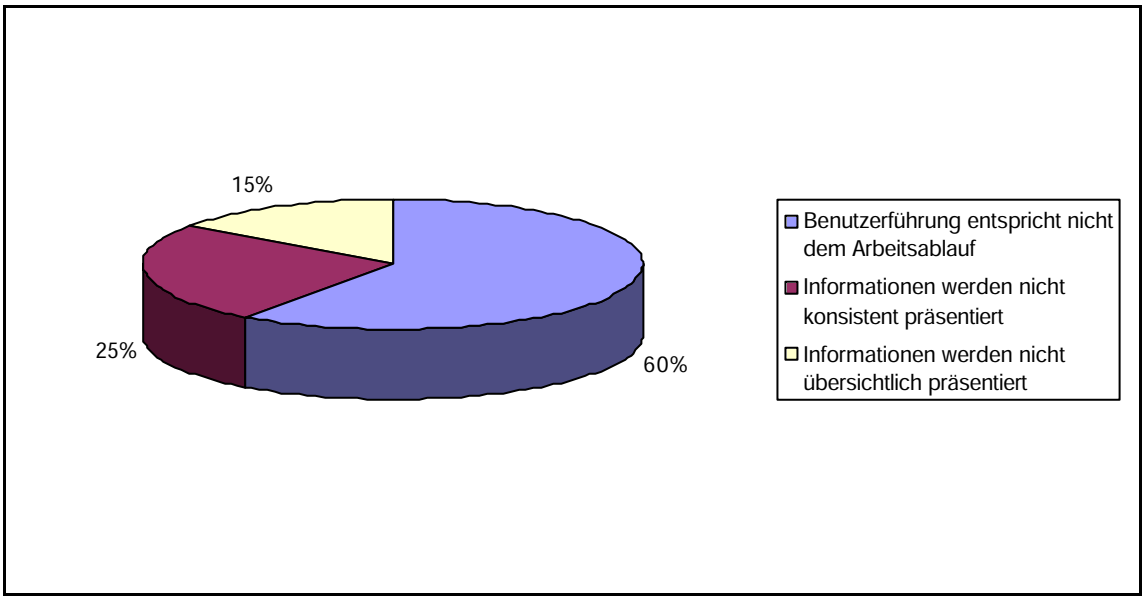


Abbildung 6.2: Ursachen für Nutzungsprobleme von Software

Die Bedeutung von Konsistenz und Übersichtlichkeit als wesentliche Qualitätsmerkmale von Nutzeroberflächen wurde schon frühzeitig erkannt. Seit der Formulierung von umfassenden, zumeist plattformspezifischen Gestaltungsrichtlinien gilt deren Einhaltung als Stand der Technik. Entsprechende Hinweise können der Fachliteratur (Microsoft, 1995b) entnommen werden und sind deshalb hier nicht Gegenstand der Untersuchung.

Deutlich komplizierter zu erreichen ist eine dem tatsächlichen Arbeitsablauf angepasste Nutzerführung. Einen wesentlichen Einfluss auf die Erreichung dieses Ziels hat die umfassende Einbeziehung der zukünftigen Anwender in die Erstellung der Lasten- und Pflichtenhefte sowie in den Entwicklungsprozess, da nur diese ausreichend präzise Kenntnisse ihrer Arbeitsabläufe haben.

Darüber hinaus muss untersucht werden, ob durch neue Konzepte der Nutzerführung eine bessere Anpassung an die Arbeitsabläufe der Anwender erreicht werden kann. Im Folgenden wird deshalb diskutiert, inwieweit Ansätze mit spezialisierten und dokumentzentrierten Nutzeroberflächen den bisher üblichen generalisierten und anwendungszentrierten Nutzeroberflächen überlegen sind. Außerdem wird das Konzept der fachspezifischen Erweiterungen von Standard-Software diskutiert.

6.2.1 Generalisierte und spezialisierte Nutzeroberflächen

Für eine dem Arbeitsablauf angepasste Benutzerführung bieten sich prinzipiell zwei gegensätzliche Wege an: generalisierte und spezialisierte Nutzeroberflächen. Auf den ersten Blick erscheinen generalisierte Nutzeroberflächen vorteilhafter, da der Anwender mit einer einzigen Anwendung eine Vielzahl von Problemen bearbeiten kann. Dieser Ansatz führt jedoch zur Entwicklung von immer komplexeren generischen Werkzeugen. Obwohl diese prinzipiell die Lösung einer großen Problemvielfalt ermöglichen, sind sie nur sehr umständlich zu benutzen, da der Nutzer selbst wissen muss, wie er seine Aufgabe mit den vom Programm zur Verfügung gestellten Mitteln bearbeiten kann. Dieser als "Mental Mapping" bezeichnete Vorgang kann sehr anspruchsvoll sein und scheitert häufig bereits daran, dass dem Nutzer die weitergehenden Möglichkeiten der benutzten Anwendung nicht bekannt sind. Ein Beispiel für generalisierte Anwendungen stellen die üblichen Standard-Textverarbeitungen dar, bei denen nach Erfahrungswerten 80% der Nutzer lediglich 20% der angebotenen Funktionalität benutzen.

Im Unterschied dazu können spezialisierte Nutzeroberflächen optimal an die Lösungsschritte eines konkreten Problems angepasst werden. Der Nutzer kann von der Anwendung Schritt für Schritt zum Ergebnis geführt werden. Die problembeschreibenden Parameter können vom Ingenieur in "seiner" Sprache spezifiziert werden. Infolgedessen kann der Arbeitsablauf sehr effizient gestaltet werden. Ungünstig ist, dass sich der Anwender für jede zusätzliche Problemklasse mit einer neuen Anwendung vertraut machen muss. Dies muss jedoch keinen gravierenden Nachteil darstellen, sofern die Nutzeroberflächen von allen eingesetzten Anwendungen konform zu plattform- und gegebenenfalls fachspezifischen Gestaltungsrichtlinien sind. Der eigentliche Nachteil spezialisierter Nutzeroberflächen ist viel gravierender und liegt in dem hohen Aufwand, der zur Erstellung einer großen Anzahl derartiger Nutzeroberflächen notwendig ist. Ein Beispiel für eine hochgradig spezialisierte Nutzeroberfläche ist ein Arbeitsblatt in einer Tabellenkalkulation oder einer Mathematik-Software. Ein derartiges Arbeitsblatt ist äußerst einfach und effizient zu benutzen, da bei einer Änderung der Parameter die Ergebnisse sofort aktualisiert werden. Diese Vorteile werden jedoch mit einer inakzeptabel niedrigen Flexibilität erkauft. Bereits bei geringfügig modifizierten Problemstellungen muss das Arbeitsblatt aufwändig angepasst oder gar völlig neu erstellt werden.

Bezogen auf das Gebiet der Tragwerksplanung sind Anwendungen zur Analyse von Stabtragwerken oder FEM-Systeme häufig mit generalisierten Nutzeroberflächen versehen. Sie gestatten innerhalb des vorgesehenen funktionalen Rahmens die Analyse beliebiger Tragwerke. Dem Anwender müssen jedoch die Konventionen und Besonderheiten der Anwendung bekannt sein. Häufig stellt die gebotene Flexibilität eher einen Nachteil dar, da viele Anwender nur ungenügende Kenntnisse der angewendeten theoretischen Grundlagen (z.B. Detaillierung der Diskretisierung, Ermittlung konsistenter Knotenlasten, Spannungsermittlung an Punktstützungen oder einspringenden Ecken) besitzen. Die daraus resultierenden Fehlbedienungen führen zu unerwarteten Ergebnissen, die in vielen Fällen irrtümlicherweise als Programmfehler interpretiert werden. Darüber hinaus erfordern generalisierte Stabtragwerks- oder FEM-Anwendungen einen sehr hohen Aufwand zur Spezifikation der benötigten Eingabeinformationen sowie bei der Interpretation der ausgegebenen Ergebnisse.

Spezialisierte Nutzeroberflächen hingegen erlauben eine wesentlich höhere Informationsdichte (Holzer, 1997), wodurch der Aufwand zur Eingabe und zur Aufbereitung der Ergebnisse drastisch reduziert werden kann. Gleichzeitig kann die Übersichtlichkeit wesentlich verbessert werden, indem die problembeschreibenden Informationen ingenieurgemäß am statischen System und nicht als diskretes Stabtragwerk oder als FEM-Modell spezifiziert werden (Abbildung 6.3).

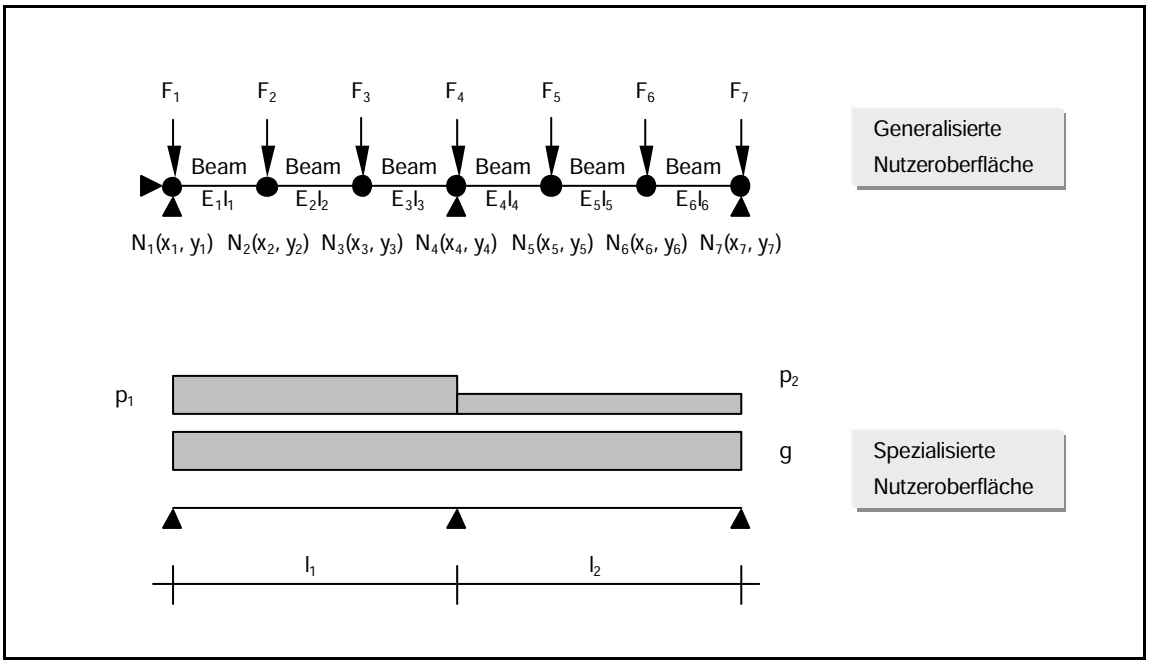


Abbildung 6.3: Generalisierte versus spezialisierte Nutzeroberfläche

Zusammenfassend wird festgestellt, dass spezialisierte Nutzeroberflächen aus der Sicht der Anwender wegen der damit erschließbaren Effizienzpotentiale unbedingt wünschenswert sind. Allerdings benötigt der Tragwerksplaner für seine vielfältigen Aufgaben eine sehr große Anzahl spezialisierter Anwendungen. Wegen des damit verbundenen Entwicklungsaufwandes wurden spezialisierte Anwendungen bislang nur selten realisiert.

6.2.2 Programm- und dokumentzentrierte Nutzeroberflächen

Bei der in Kapitel 6.1.1 beschriebenen, gegenwärtig typischen Arbeitsweise wird das zu untersuchende Tragwerk vom Tragwerksplaner in überschaubare Teiltragwerke zerlegt. Für jede Position werden die Schnittgrößen getrennt ermittelt und damit die Bauteile bemessen sowie deren Tragsicherheit und Gebrauchstauglichkeit nachgewiesen. Anschließend werden die Ergebnisse im Tragwerksbericht dokumentiert. Im Regelfall wird für jeden Schritt ein individuelles Programm verwendet. Insgesamt sind zur Bearbeitung eines Bauteils drei verschiedenartige Anwendungen erforderlich, von denen jede über eine eigene Nutzeroberfläche und eine eigene Datenhaltung verfügen kann.

Derartige Oberflächen mit herkömmlicher Nutzerführung werden als anwendungszentriert bezeichnet. Aufgrund der häufig inkonsistent gestalteten Nutzeroberflächen ist anfänglich ein hoher Einarbeitungsaufwand erforderlich. Wesentlich störender machen sich bei der täglichen Arbeit die unbefriedigenden Interaktionsmöglichkeiten zwischen den beteiligten Anwendungen bemerkbar. Infolgedessen müssen die Ergebnisse aufwändig in die nachfolgend eingesetzten Anwendungen übernommen werden. Beim Eintreten von Änderungen müssen die einzelnen Programme erneut aufgerufen werden, die Berechnung mit den geänderten Parametern gestartet und anschließend die Daten über Export- und Importfilter ausgetauscht werden (Abbildung 6.4). Die erzeugten Ergebnisse werden durch den Tragwerksplaner kommentiert und formatiert. Bei einem erneuten Import von Daten werden entweder die bestehenden Inhalte überschrieben oder die doppelt vorhandenen Informationen müssen manuell entfernt werden. Damit gehen sämtliche zwischenzeitlich vorgenommenen Kommentierungen oder Formatierungen verloren und müssen aufwändig wiederholt werden.

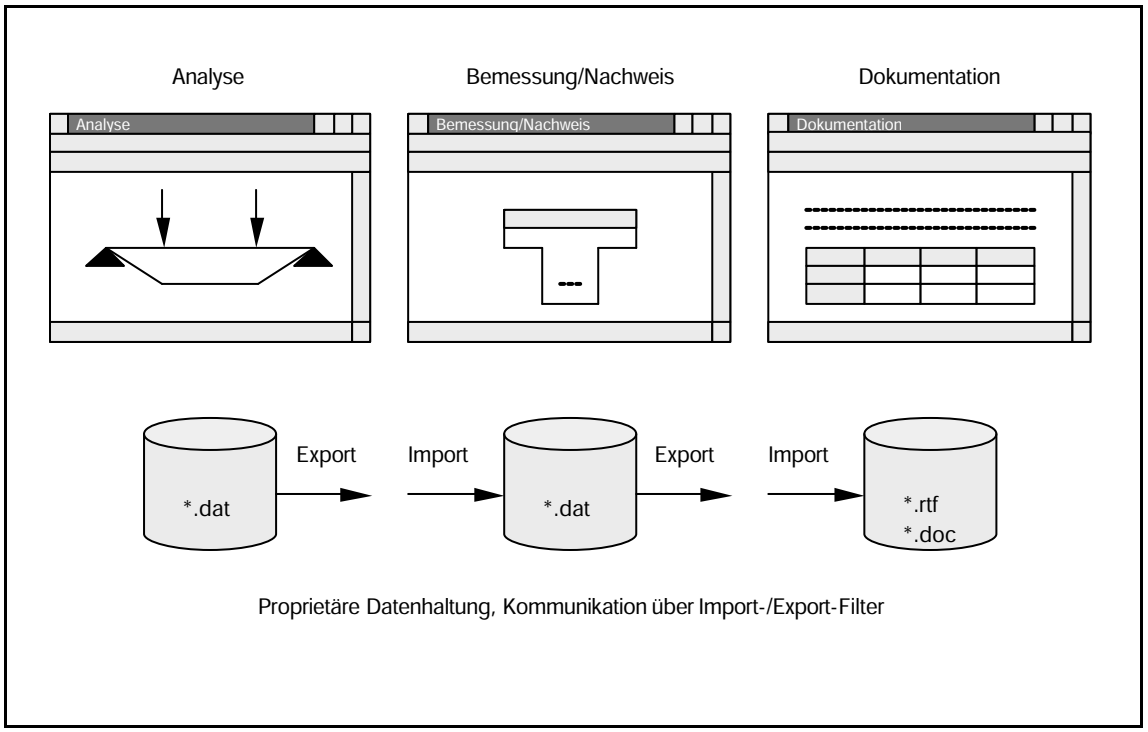


Abbildung 6.4: Programmzentrierte Nutzeroberfläche

Die drei Arbeitsschritte Analyse, Bemessung und Nachweis eines Bauteils sowie deren Dokumentation gehören logisch zusammen und sollten zweckmäßigerweise für jeweils eine Problemstellung zusammengefasst werden. Ein vielversprechender Ansatz zur Lösung dieser Problematik wird seit geraumer Zeit in sogenannten Verbunddokumenten gesehen (Beucke, 1993), (Beucke, 1996).

Oberflächen mit einer derartigen Nutzerführung werden als dokumentzentriert bezeichnet. Durch die gemeinsame Darstellung aller relevanten Informationen in einem Dokument werden Übersichtlichkeit und Arbeitsfluss wesentlich verbessert. Aufgrund der gemeinsamen Datenhaltung können die bei anwendungszentrierten Nutzungs- und Datenmodellen erforderlichen arbeitsaufwändigen Export- und Importvorgänge komplett entfallen, da eine Integration der beteiligten Daten bei diesem Konzept inhärent ist (Abbildung 6.5). Trotz der beschriebenen Vorteile wurden dokumentzentrierte Nutzeroberflächen von Software für die Tragwerksplanung bislang nur ansatzweise realisiert.

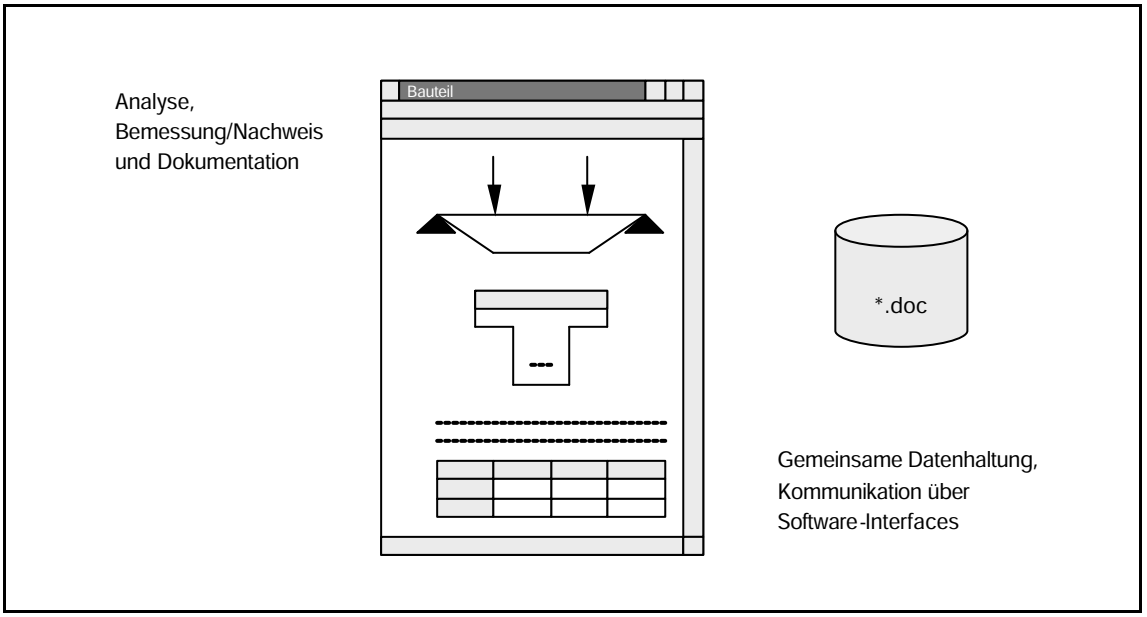


Abbildung 6.5: Dokumentzentrierte Nutzeroberfläche

6.2.3 Integration fachspezifischer Funktionalität in Standard-Software

Eine weitere Möglichkeit zur ergonomischen Gestaltung von Nutzeroberflächen stellt der möglichst umfassende Einsatz von Standard-Software dar. Standard-Software zeichnet sich im Allgemeinen durch eine hohe Benutzerfreundlichkeit aus, da ein grundlegendes Verständnis der Bedienung von Standard-Software mittlerweile vorausgesetzt werden kann. Dabei muss jedoch darauf geachtet werden, dass die in Kapitel 6.2.1 beschriebenen Nachteile generalisierter Nutzeroberflächen vermieden werden. Dies kann erreicht werden, indem Standard-Software um fachspezifische Funktionalität erweitert und dadurch mit einer spezialisierten Nutzeroberfläche versehen wird. Durch die Kombination der intuitiven Bedienbarkeit infolge der standardisierten Grundfunktionalität mit der Effizienz der spezialisierten fachspezifischen Funktionalität kann eine drastisch verkürzte Einarbeitungszeit bei deutlich gesteigerter Arbeitsproduktivität erreicht werden.

6.2.4 Einbeziehung der Anwender in den Entwicklungsprozess

Wie bereits festgestellt wurde, setzt eine dem Arbeitsablauf entsprechende Benutzerführung von Tragwerksplanungs-Software voraus, dass die zukünftigen Anwender möglichst umfassend in die Erarbeitung der Lasten- und Pflichtenhefte sowie die Software-Entwicklung eingebunden werden, da nur diese über ausreichend präzise Kenntnisse ihrer Arbeitsabläufe verfügen.

Der traditionelle Weg zur Erfüllung dieser Anforderung bestand darin, dass ein großer Teil der in der Tragwerksplanung eingesetzten Software von Bauingenieuren selbst entwickelt wurde. Mit der Etablierung von grafischen Nutzeroberflächen und dem damit verbundenen Paradigmenwechsel zur objektorientierten Programmierung wurde in der Software-Entwicklung eine Stufe der Komplexität erreicht, die nur noch durch professionelle Software-Entwickler mit fundierter theoretischer Ausbildung und ausreichender Berufspraxis beherrscht werden kann. Bauingenieure, die weiterhin in ihrem angestammten Aufgabengebiet arbeiten und die Entwicklung der Informatik nur als Hobby verfolgen, können über die geforderte software-technische Professionalität nur noch mit Einschränkungen verfügen. Die Entwicklung von Software für die Tragwerksplanung kann deshalb nicht mehr ausschließlich durch Bauingenieure selbst erfolgen, da andernfalls die Produkte unter informationstechnischen Aspekten nicht mehr dem Stand der Technik entsprechen würden.

Damit die Entwicklung von Tragwerksplanungs-Software durch praktisch tätige Bauingenieure wieder möglich wird, muss vor allen Dingen die Komplexität der Software-Entwicklung reduziert werden. Offensichtlich kann dies nur durch eine zweckmäßige Aufgabenteilung zwischen Informatikern, Bauingenieuren mit verstärkter Ausbildung in der Informatik (Bauinformatikern) sowie den praktisch tätigen Bauingenieuren (z.B. Tragwerksplanern) erreicht werden.

In diesem Szenario sind Informatiker als klassische Software-Entwickler für die Evaluierung und Selektion von Basistechnologien, den Entwurf von Systemarchitekturen sowie die Entwicklung von Standardfunktionalität und sonstiger nicht fachspezifischer Funktionalität verantwortlich. Zu den Aufgaben der Bauinformatiker gehört es, die fachspezifische Funktionalität zu implementieren und an die konkreten Erfordernisse der Tragwerksplaner anzupassen. Darüber hinaus nimmt der Bauinformatiker eine Schlüsselstellung ein, indem er einerseits die Tragwerksplaner bei der Formulierung ihrer Produktanforderungen berät und andererseits die Informatiker bei der Erarbeitung einer ingenieurgemäßen Lösung unterstützt. Den Tragwerksplanern obliegt es, praxisgerechte Vorgaben für die Erstellung der Lasten- und Pflichtenhefte zu erarbeiten und den Fortgang der Entwicklung aus ihrer Perspektive zu überwachen. Außerdem sind sie für die Anpassung der Software im Detail verantwortlich.

Darüber hinaus ist es wichtig, dass die Komplexität des Software-Entwicklungsprozesses auf mehreren Ebenen gekapselt wird, so dass die komplexen informationstechnischen, aber nicht-fachspezifischen Arbeitsschritte durch Informatiker erbracht werden können, wohingegen sich Bauinformatiker und insbesondere Tragwerksplaner auf die Bearbeitung ihrer fachspezifischen Aufgaben konzentrieren können.

Die geforderte stufenweise Kapselung der Komplexität kann unter anderem durch die Anwendung der bereits diskutierten dokumentzentrierten Software-Architektur erreicht werden. Damit wird eine Aufteilung des Entwicklungsprozesses in Container-Anwendung, Steuerelemente (Controls) und Scripte möglich. Die größte informationstechnische Herausforderung stellt dabei die Entwicklung der generischen Container-Anwendung dar. Die Entwicklung kann durch Informatiker erfolgen, da dazu keine fachspezifischen Kenntnisse erforderlich sind. Die Container-Anwendung wird durch die Aufnahme von fachspezifischen Steuerelementen für das gewünschte Anwendungsgebiet spezialisiert. Die Entwicklung der fachspezifischen Steuerelemente stellt eine mittlere Komplexitätsstufe dar, die ein gutes Verständnis sowohl der informationstechnischen als auch der fachspezifischen Belange erfordert. Für diese Aufgabe sind Bauinformatiker prädestiniert. Die bereitgestellten Steuerelemente müssen schließlich zu aufgabenbezogenen Dokumenten zusammengefasst werden. Dabei muss eine zweckmäßige Dokumentstruktur festgelegt und durch Scripting der Informationsfluss zwischen den Steuerelementen definiert werden. Dieser Vorgang stellt nur geringe informationstechnische Anforderungen, da keinerlei eigene Funktionalität implementiert werden muss, sondern nur die vorgefertigten Steuerelemente als Black-Box-Bausteine unter fachlichen Gesichtspunkten verknüpft werden müssen. Infolge des geringen informationstechnischen Anteils können solche Aufgaben auch durch interessierte, praktisch tätige Tragwerksplaner realisiert werden (Abbildung 6.6).

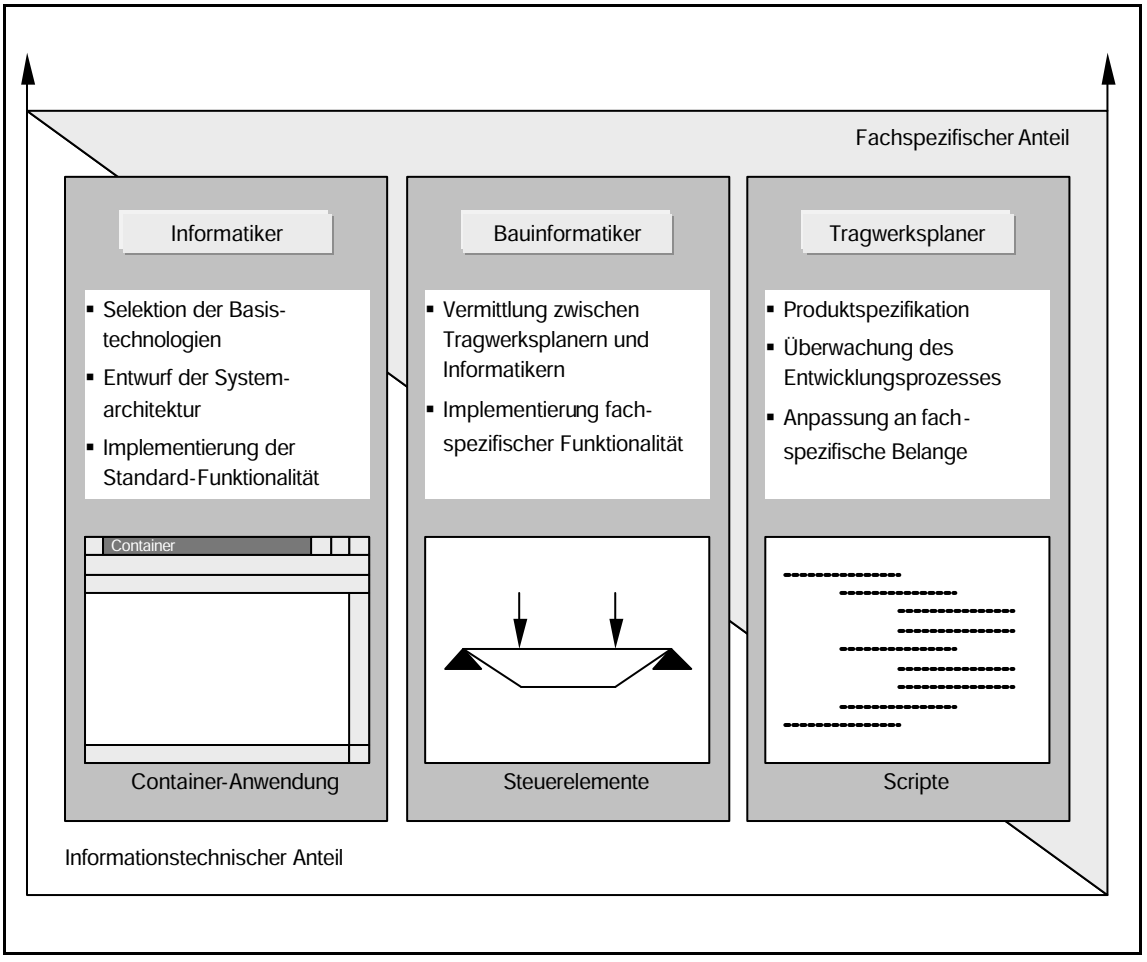


Abbildung 6.6: Arbeitsteilung zwischen Informatikern, Bauinformatikern und Tragwerksplanern

6.3 Integration von Tragwerksplanungs-Software

Unter Integration wird verstanden, dass Anwendungen untereinander Informationen austauschen können beziehungsweise auf einem einheitlichen Datenbestand operieren. Adäquate Integrationsmöglichkeiten können wesentlich zu ingenieurgemäßen Arbeitsumgebungen beitragen, da sie es dem Tragwerksplaner ermöglichen, auf die bereits in digitaler Form vorliegenden Informationen zurückzugreifen. Dadurch kann sowohl eine Verkürzung der Bearbeitungsdauer ermöglicht als auch die Qualität der erzeugten Unterlagen erhöht werden.

6.3.1 Klassifizierung von Integrationsansätzen

Prinzipiell können zwei verschiedene Arten von Integrationsansätzen unterschieden werden:

- Integrationsansätze mit Interpretation der Semantik
- Integrationsansätze ohne Interpretation der Semantik

Die erste Möglichkeit verkörpert den klassischen Integrationsansatz. In diesem Szenario muss vorab ein einheitliches Informationsmodell definiert werden. Auf dieser Grundlage können Integrationswerkzeuge implementiert werden, welche die Semantik der Informationen selbständig interpretieren können, ohne dass dazu ein Eingriff des Anwenders erforderlich ist.

Alle Integrationsansätze, die auf einer Interpretation der Semantik der Informationen basieren, erfordern einen sehr hohen Aufwand an Forschung, Entwicklung und Koordinierung, da entweder das Format der Speicherung von Informationen in Dateien oder die Kommunikationsmöglichkeiten zwischen den verschiedenen Anwendungen mittels Software-Interfaces standardisiert werden müssen (Schneider, 1999). Die Realisierung eines derartigen Integrationsansatzes ist seit Jahren Gegenstand der internationalen Norm ISO 10303 "Standard for the Exchange of Product Model Data (STEP)". Die dabei entwickelten Grundlagen wurden von der Industry Alliance for Interoperability aufgegriffen und mit den sogenannten "Industry Foundation Classes (IFC)" an die Bedürfnisse des Bauwesens angepasst. Darüber hinaus wurden zahlreiche Forschungsarbeiten an Universitäten und in Unternehmen durchgeführt, die ebenfalls eine automatische Interpretation der Semantik der Informationen zum Ziel haben.

Der größte Nachteil der auf einer Interpretation der Semantik basierenden Integrationsansätze liegt in ihrer Komplexität. Bereits die an einem Bauprojekt beteiligten Mitarbeiter haben völlig unterschiedliche Anforderungen an die Detaillierung und Strukturierung der Informationen. Hinzu kommen traditionell unterschiedliche nationale Gepflogenheiten. Infolgedessen haben sich alle Bestrebungen zur branchen- und weltweiten Standardisierung als sehr schwierig und langwierig herausgestellt. Obwohl bereits partiell eine Interoperabilität im Bauwesen demonstriert werden konnte, steht die Erreichung des ursprünglichen Ziels eines umfassenden und allgemein gültigen Integrationskonzeptes für das Bauwesen noch aus und wird vermutlich auch auf absehbare Zeit nicht erreicht werden (Bittrich, 1998).

Im Unterschied zum zuvor diskutierten Integrationsansatz mit Interpretation der Semantik verzichtet der zweite mögliche Ansatz bewusst auf eine automatische Interpretation der Semantik der Informationen (Huhnt, 1997), (Bittrich, 1997), (Hinz, 1998), (Schneider, 1999). Stattdessen ist der Anwender selbst für die Auswahl der relevanten Informationen und die korrekte Interpretation von deren Semantik verantwortlich. Der Informationsfluss wird vom Anwender zur Bearbeitungszeit festgelegt und vom Integrationswerkzeug protokolliert. Zu diesem Zweck muss der Anwender zunächst die Entnahmestelle und die Einfügestelle der Information spezifizieren. Danach kann die Information selbst vom Integrationswerkzeug ohne weiteren Eingriff des Anwenders übertragen werden. Mit Hilfe des vom Integrationswerkzeug protokollierten Informationsflusses können durch Änderungen hervorgerufene Inkonsistenzen erkannt und auf Veranlassung des Anwenders automatisch aufgehoben werden.

Der offensichtliche Vorteil von Integrationskonzepten ohne automatische Interpretation der Semantik der Informationen liegt darin, dass nicht vorab ein vollständiges Informationsmodell definiert werden muss. Da im Wesentlichen lediglich eine Methodik zur Identifizierung der Informationen und zur Protokollierung des Informationsflusses benötigt wird, ist auch der Realisierungsaufwand für ein derartiges Integrationskonzept um Größenordnungen geringer als bei Integrationskonzepten mit automatischer Interpretation der Semantik. Hinzu kommt der Vorteil, jederzeit Erweiterungen vornehmen zu können, die ursprünglich nicht vorgesehen waren.

In Anbetracht der bisher ungelösten Probleme von Integrationskonzepten, die auf einer Interpretation der Semantik der Informationen beruhen, sollte auch untersucht werden, inwieweit Integrationskonzepte eine Alternative darstellen, die auf eine Interpretation der Semantik der Informationen verzichten. Zum gegenwärtigen Zeitpunkt erscheint es zweckmäßig, Integrationsmöglichkeiten stets nur so umfassend wie zur effizienten Bearbeitung einer konkreten Aufgabenstellung nötig, keinesfalls aber so umfassend wie theoretisch möglich zu gestalten.

6.3.2 Informationsfluss in der Tragwerksplanung

Eine Analyse des Informationsflusses bildet zwangsläufig die Grundlage von integrierten Anwendungen. Im Folgenden wird der Informationsfluss während der Tragwerksplanung in vereinfachter Form dargestellt (Abbildung 6.7), wobei auf die in Kapitel 6.1.1 definierten Phasen Bezug genommen wird.

1. Entwurf des Tragwerkes

   Der Tragwerksplaner kann auf die CAD-Daten des Architekten oder des Objektplaners zurückgreifen. In der Praxis findet gegenwärtig lediglich ein Austausch von Zeichnungsdateien auf dem Niveau zweidimensionaler geometrischer Grundelemente statt. Wegen der systematischen Unzulänglichkeiten des Datenaustausches zwischen verschiedenen CAD-Systemen ist in vielen Fällen ein hoher Nachbereitungsaufwand erforderlich. Häufig müssen die Informationen neu strukturiert werden. Im Anschluss daran werden nicht benötigte Informationen entfernt und fehlende Informationen hinzugefügt. Das Ergebnis dieser Phase ist ein geometrisches Modell.

2. Idealisierung von Tragwerk und Einwirkungen

   Eine direkte Überführung des geometrischen Modells in ein Tragwerksmodell ist nur mit großen Einschränkungen möglich. Die dazu benötigten Schritte der geometrischen Idealisierung, der Identifikation der Teiltragwerke, die Dimensionsreduzierung sowie die Bestimmung von Auflagerachsen und Randbedingungen stellen nicht triviale Aufgaben dar. Darüber hinaus ist eine Informationsübernahme für alle Bauteile gar nicht erforderlich, da für den Tragwerksplaner nur die maßgebenden Bauteile relevant sind. Der Prozess der Idealisierung kann bislang nur ansatzweise durch Software unterstützt werden. Folglich ist ein sehr hoher Aufwand zur manuellen Verdichtung der Informationen erforderlich. Unter diesen Umständen bringt eine Übernahme der geometrischen und topologischen Informationen aus den Unterlagen des Objektplaners nur bei sehr komplexen Bauteilen deutliche Vorteile. Bei Bauteilen geringer oder mittlerer Komplexität ist eine Neueingabe der Informationen häufig schneller und deshalb in der Praxis üblich.

3. Ermittlung der Schnittgrößen

   Die Informationen des Tragwerkmodells beschreiben eine Problemstellung im Regelfall eindeutig. Prinzipiell kann ein Tragwerksmodell ohne Eingriff des Anwenders in die benötigten numerischen Modelle transformiert und analysiert werden. Die Ergebnisse werden zweckmäßigerweise in einem separaten Ergebnismodell abgelegt. Die in der Regel sehr umfangreichen Informationen des numerischen Modells (z.B. Finite-Element-Diskretisierung) müssen nach außen nicht sichtbar werden. Damit kann eine Integration innerhalb der Phase der Schnittgrößenermittlung wesentlich zu einem ingenieurgemäßen Arbeitsablauf beitragen (Holzer, 1997).

4. Bemessung/Nachweis

   Die Informationen des Ergebnismodells müssen nur an den bemessungsrelevanten Punkten übernommen werden, die auszutauschende Informationsmenge ist somit begrenzt. Sofern die Anwendungen zur Schnittgrößenermittlung und zur Bemessung beziehungsweise Nachweisführung zu einer Einheit zusammengefasst werden, sind keine speziellen Maßnahmen zur Integration erforderlich. Bislang nur unzureichend integriert ist jedoch die Möglichkeit der Dokumentation in Textverarbeitungs-Software.

5. Konstruktive Durchbildung

   Zur konstruktiven Durchbildung müssen ausgewählte Informationen aus der Phase Bemessung/Nachweis übernommen werden. Darüber hinaus werden zusätzliche Informationen benötigt, die auf Erfahrungswerten des Tragwerksplaners sowie Normen und Richtlinien basieren. Diese Phase kann gegenwärtig nicht sinnvoll automatisiert werden. Während der konstruktiven Durchbildung werden die bisherigen Arbeitsschritte durch den Tragwerksplaner implizit kontrolliert, so dass offensichtliche Fehler in der Regel erkannt werden. Eine vollständige Automatisierung erscheint deshalb auch zukünftig nicht empfehlenswert. Als Ergebnis der konstruktiven Durchbildung werden mit CAD Konstruktionspläne erstellt. Diese Pläne stellen Detailpläne dar. Eine Informationsübernahme aus diesen Plänen ist nur für besondere Aufgaben erforderlich, weshalb eine vollständige Integration nicht notwendig ist.

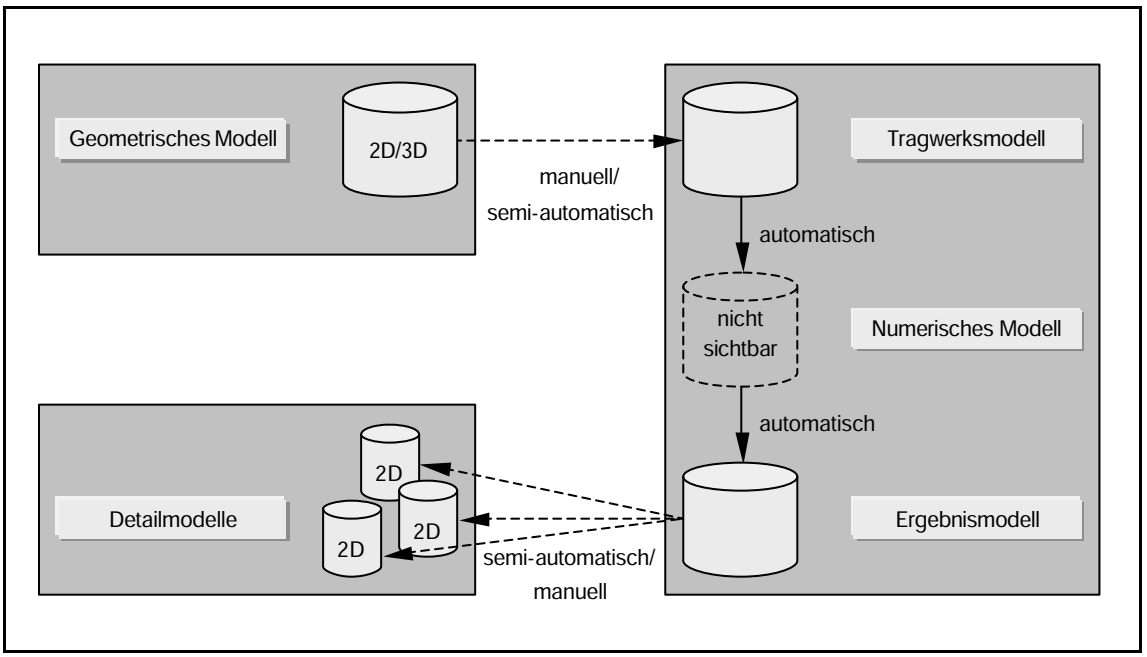


Abbildung 6.7: Informationsfluss in der Tragwerksplanung

6.3.3 Integration von CAD-Software

Die Zielstellung dieser Arbeit liegt vordergründig auf der besseren Unterstützung des Tragwerksplaners bei Routineaufgaben. Für diese Aufgaben würde eine automatische Generierung des Tragwerkmodells aus dem geometrischen Modell nur geringe Vorteile bringen, die hauptsächlich in einem leichteren Erkennen von Änderungen liegen würden.

Die Durchführung der Tragwerksplanung wird in Form des Tragwerksberichtes und von Konstruktionszeichnungen dokumentiert. Eine Unterstützung des Konstrukteurs bei der Erstellung der Konstruktionszeichnungen stellt eine deutliche Arbeitserleichterung dar und wird teilweise bereits durch kommerzielle Systeme unterstützt.

Die Aspekte einer Anbindung von CAD-Software bedürfen einer eigenständigen Untersuchung und werden hier nicht weiter verfolgt.

6.3.4 Unterstützung des Lastabtrages

Neben den offensichtlichen Vorteilen der Integration von Tragwerksmodellen, numerischen Modellen und Ergebnismodellen kommt dem Zusammenspiel der einzelnen Teiltragwerke zentrale Bedeutung zu. Wie in Kapitel 6.1.2 erläutert, werden Tragwerke im Regelfall als dimensionsreduzierte Teiltragwerke modelliert. Dadurch wird einerseits die Analyse der Teiltragwerke wesentlich vereinfacht. Auf der anderen Seite muss der Lastabtrag zwischen den einzelnen Teiltragwerken durch den Ingenieur explizit vorgenommen werden. Da sowohl der Prozess der Bauplanung als Ganzes als auch der Prozess der Tragwerksplanung iterative Prozesse sind, stellt die kontinuierliche Aktualisierung des Tragwerksmodells für den Tragwerksplaner eine zeitaufwändige und fehleranfällige Aufgabe dar. Sofern die Änderungen zu verringerten Einwirkungen oder zu einer Erhöhung des Bauteilwiderstandes führen, wird häufig unter bewusster Inkaufnahme von Inkonsistenzen auf eine Aktualisierung des Tragwerkmodells verzichtet.

Als Ergänzung zu diesem pragmatischen Lösungsansatz wurden zahlreiche Versuche unternommen, den Lastabtrag durch semantische Tragwerksmodelle automatisch zu verwalten (Molkenthin, 1994a) (Rüppel, 1994), (Brettschneider, 1998). Semantische Tragwerksmodelle stellen Integrationsansätze mit einer Interpretation der Semantik der Informationen dar. Die Mehrzahl der publizierten Ansätze basiert auf einer Tragwerksmodellierung mit einer endlichen Anzahl von Bauteil- und Kopplungstypen. Damit ist es für spezielle Aufgabenstellungen möglich, entsprechend der spezifizierten Parameter und definierten Regeln den Lastabtrag automatisch zu generieren. In (Hinz, 1998) wird anhand eines konkreten Beispiels nachgewiesen, dass dies im allgemeinen Fall wegen der Unvollständigkeit der Modelle nicht möglich ist. Weiterhin wird gezeigt, dass die Modellierung mit semantischen Tragwerksmodellen nicht immer eindeutig möglich ist und dass der Aufwand zur Erstellung solcher Modelle mit zunehmender Komplexität der Tragstruktur stark ansteigt. Als Alternative wird ein verallgemeinertes Tragwerksmodell vorgeschlagen, das auf eine automatische Interpretation der Semantik der Informationen verzichtet. Stattdessen werden die Verknüpfungen zwischen den Bauteilen vom Tragwerksplaner während der Bearbeitung festgelegt und durch die Software protokolliert. Damit können Informationen semi-automatisch übernommen werden. Inkonsistenzen zwischen den Teiltragwerken können durch Auswertung des Verknüpfungsgraphen erkannt und auf Wunsch aufgehoben werden.

Wie alle semantikfreien Integrationskonzepte ist auch das Konzept der Protokollierung von Verknüpfungen mit Nachteilen behaftet (Bittrich, 1997). Solange nur Eigenschaften der verknüpften Teiltragwerke geändert werden, können alle Änderungen zuverlässig erkannt und Inkonsistenzen aufgehoben werden. Durch strukturelle Änderungen (z.B. Entfernen, Einfügen, Zusammenfassung und Trennung von Teiltragwerken) wird jedoch der Verknüpfungsgraph modifiziert und muss zumindest partiell vom Bearbeiter erneut erstellt werden. Im Unterschied dazu kann ein semantisches Tragwerksmodell den Lastabtrag jederzeit neu generieren, ohne dass dazu ein Eingriff des Nutzers nötig wäre.

Dennoch stellt das Konzept einer semantikfreien Integration eine ingenieurgemäße Lösung dar, da der Ingenieur bei Routinetätigkeiten unterstützt wird, aber gleichzeitig die volle Freiheit über die Wahl eines geeigneten Tragwerkmodells behält und sich nicht einem starren, vordefinierten Regelwerk unterordnen muss.

Es stellt sich weiterhin die Frage, inwieweit ein automatischer und exakter Lastabtrag überhaupt zweckmäßig ist. Entsprechend der normierten Regeln ist stets die am ungünstigsten wirkende Laststellung zu betrachten. Bereits bei Aufgabenstellungen mittlerer Komplexität kann die ungünstigste Laststellung vom Tragwerksplaner nicht mehr a priori identifiziert werden. Diese Aufgabe kann nur noch rechnergestützt durch Berechnung aller möglichen Kombinationen gelöst werden. Diese Problematik wird durch den Übergang von deterministischen zu semi-probabilistischen Sicherheitskonzepten weiter verschärft. Zum einen müssen wegen unterschiedlicher Teilsicherheitsbeiwerte die Lastanteile getrennt nach ständigen und veränderlichen sowie günstigen und ungünstigen Anteilen ermittelt werden. Infolge unterschiedlicher Kombinationsbeiwerte müssten genau genommen auch die Herkunft aller Lastanteile (Wind, Schnee, sonstige Verkehrslasten) beziehungsweise die Nutzungen der Räume berücksichtigt werden.

Da derartig komplexe Tragwerksmodelle nur rechnergestützt bearbeitet werden können und nicht mehr durch Handrechnungen zu überprüfen sind, wird folgende Vereinfachung vorgeschlagen: Für ein konkretes Bauteil werden ausgehend von den charakteristischen Einwirkungen entsprechend der genormten Vorschriften die Schnittgrößen im Grenzzustand der Tragfähigkeit ermittelt. Nach den jeweiligen Erfordernissen können auch ausgewählte Schnittgrößen in den verschiedenen Grenzzuständen der Gebrauchstauglichkeit ermittelt werden. Mit den ermittelten Schnittgrößen können die Bemessungen und Nachweise in den Grenzzuständen der Tragfähigkeit und der Gebrauchstauglichkeit durchgeführt werden.

Um den Lastabtrag auf die darunter liegenden Positionen vorzunehmen, müssen die Minimal- und Maximalwerte der Auflagerreaktionen bestimmt werden. Unter der Annahme, dass die minimale Reaktion vom ständigen Lastanteil stammt, und die Differenz aus maximaler und minimaler Reaktion vom veränderlichen Lastanteil hervorgerufen wird, können daraus mit Hilfe der folgenden Gleichungen die charakteristischen Einwirkungen für das darunter liegende Bauteil ermittelt werden.

$$G_k = \frac{\text{Min } S_d}{\gamma_G} \quad \text{und} \quad g_k = \frac{\text{Min } s_d}{\gamma_G}$$

$$Q_k = \frac{\text{Max } S_d - \text{Min } S_d}{\gamma_Q} \quad \text{und} \quad q_k = \frac{\text{Max } s_d - \text{Min } s_d}{\gamma_Q}$$

G_k, g_k	charakteristischer Wert der ständigen Einwirkung
Q_k, q_k	charakteristischer Wert der veränderlichen Einwirkung
S_d, s_d	Bemessungswert der Einwirkungen
γ_G	Teilsicherheitsbeiwert für ständige Einwirkungen
γ_Q	Teilsicherheitsbeiwert für veränderliche Einwirkungen

Diese Annahmen erscheinen im Verhältnis zu den sonstigen Vereinfachungen und Ungenauigkeiten für übliche Hochbauten tragbar und erlauben gemeinsam mit dem Konzept der Protokollierung von Verknüpfungen eine ingenieurgemäße Lösung des Lastabtrages.

Kapitel 7
Software-technische Konzeption

Im vorangegangenen Kapitel wurde aus der Sicht der Tragwerksplaner herausgearbeitet, nach welchen Prinzipien eine ingenieurgemäße Tragwerksplanungs-Software gestaltet werden sollte. Eine wesentliche Erkenntnis dabei war, dass innovative Konzepte zur ingenieurgemäßen Gestaltung von Tragwerksplanungs-Software bislang nur vereinzelt umgesetzt werden konnten, da sie mit einem unverhältnismäßig hohen Entwicklungsaufwand verbunden sind. In diesem Kapitel wird deshalb der Versuch unternommen, von Beginn an durch gezielte Maßnahmen und eine entsprechende software-technische Konzeption den zur Realisierung derartiger Tragwerksplanungs-Software erforderlichen Aufwand zu minimieren. Dabei wird insbesondere auf die folgenden Ansätze eingegangen:

- Erhöhung des Wiederverwendungsgrades
- Erweiterbarkeit existierender Systeme

Weiterhin wird die software-technische Konzeption auf der Grundlage von Verbunddokumenten als erweiterbare Nutzeroberfläche und als flexibler Datenspeicher erläutert.

7.1 Erhöhung des Wiederverwendungsgrades

Die Wiederverwendung von vorgefertigten Software-Bausteinen ist eine besonders zweckmäßige Möglichkeit zur Realisierung einer produktivitätssteigernden Arbeitsteilung in der Software-Industrie. Die Kostenstruktur von Software wird durch sehr hohe Entwicklungskosten dominiert. Im Vergleich dazu sind die Kosten für Herstellung und Vermarktung eher marginal. Das heißt, dass die Grenzkosten für jede weitere verkaufte Lizenz ebenfalls marginal sind. Um diesen Skaleneffekt ausnutzen zu können, müssen Software-Hersteller einen möglichst hohen Teil der implementierten Funktionalität anderen Herstellern verfügbar machen. Aus diesen betriebswirtschaftlichen Zwängen heraus wird mittelfristig eine eigene Software-Entwicklung nur noch innerhalb der Kernkompetenzen wirtschaftlich vertretbar sein, alle übrigen Software-Bausteine werden zu geringeren Kosten von spezialisierten Anbietern dazu gekauft.

Von den Konzepten zur Wiederverwendung, die bereits Eingang in die industrielle Praxis gefunden haben, stellt die komponentenorientierte Software-Technik den gegenwärtig leistungsfähigsten Ansatz dar. Um den Wiederverwendungsgrad durch Einsatz von Software-Komponenten zu erhöhen, muss unter Berücksichtigung der in Kapitel 5.2 diskutierten Überlegungen der durch Software-Komponenten realisierte Anteil an der Implementierung gesteigert werden. Zu diesem Zweck können drei verschiedene Ansätze verfolgt werden:

- Ausgliederung problemspezifischer Funktionalität in Software-Komponenten ohne eigene Nutzeroberfläche
- Ausgliederung problemspezifischer Funktionalität in Software-Komponenten mit eigener Nutzeroberfläche
- Nutzung von durch Standard-Software bereitgestellter Funktionalität

7.1.1 Software-Komponenten ohne eigene Nutzeroberfläche

Der Einsatz von Software-Komponenten ohne eigene Nutzeroberfläche ist am einfachsten zu realisieren. Derartige Software-Komponenten sind für alle Aufgaben geeignet, die keine direkten Nutzereingriffe erfordern und weitgehend autonom, das heißt ohne umfangreiche Abhängigkeiten zu anderen Programmbestandteilen, abgearbeitet werden können.

Im Bereich der Tragwerksplanung bietet sich somit insbesondere Funktionalität zur Durchführung von Stabtragwerks- und FEM-Analysen zur Ausgliederung in Software-Komponenten an. Derartige Software-Komponenten weisen zwangsläufig eine sehr grobe Granularität auf. Infolgedessen ist es im Regelfall sinnvoll, Funktionalität für bestimmte wiederkehrende Aufgaben wie beispielsweise die Vernetzung oder das Lösen von Gleichungssystemen zu generalisieren und als untergeordnete Software-Komponenten auszugliedern (Röder, 1998a), (Röder, 1998b).

Mit dem Einsatz der zuvor beschriebenen Software-Komponenten ohne eigene Nutzeroberfläche ist es möglich, die sogenannte Kernfunktionalität in mehreren Anwendungen wiederzuverwenden. Da die Kernfunktionalität zu den langlebigen Software-Bausteinen gehört, ist deren Ausgliederung in Software-Komponenten unter ökonomischen Gesichtspunkten besonders attraktiv und wird auch schon vereinzelt praktiziert. Erfahrungsgemäß ist der Implementierungsaufwand für die Kernfunktionalität jedoch wesentlich niedriger als der zur Entwicklung der Nutzeroberflächen. Beispielsweise werden in (Hinz, 1998) für die realisierten Prototypen Verhältnisse von 1:3 bis 1:10 publiziert. Bei einer Umsetzung in kommerzielle Produkte muss tendenziell mit noch ungünstigeren Verhältnissen gerechnet werden. Da der Großteil des Implementierungsaufwandes für die Entwicklung der Nutzeroberfläche benötigt wird, kann durch den ausschließlichen Einsatz von Software-Komponenten ohne eigene Nutzeroberfläche auch der Wiederverwendungsgrad nur in sehr begrenztem Umfang erhöht werden.

7.1.2 Software-Komponenten mit eigener Nutzeroberfläche

Um den Komponentenanteil über das bisher erreichte Niveau hinaus zu steigern, muss die bisher nicht betrachtete Implementierung der Nutzeroberfläche zumindest teilweise wiederverwendet werden. Dazu müssen auch Möglichkeiten genutzt werden, um Funktionalität in Software-Komponenten mit eigener Nutzeroberfläche auszugliedern. Da die einzelnen Software-Komponenten mit eigener Nutzeroberfläche später zu einer konsistent und übersichtlich gestalteten Anwendung zusammengeführt werden müssen, stellt diese Vorgehensweise wesentlich höhere Anforderungen als die bisher praktizierten Ansätze zur Ausgliederung der Kernfunktionalität.

Eine Möglichkeit zur Realisierung von Software-Komponenten mit eigener Nutzeroberfläche stellen die sogenannten Controls oder Steuerelemente (Abbildung 7.1) dar, die bereits in Kapitel 4.2 vorgestellt wurden. Steuerelemente eignen sich insbesondere zur Darstellung von häufig in ähnlicher Form wiederkehrenden Dokumentteilen. Bezogen auf das Gebiet der Tragwerksplanung kommen Steuerelemente beispielsweise zur Visualisierung von statischen Systemen und Schnittgrößenverläufen sowie zur Durchführung von konkreten Bauteilbemessungen und -nachweisen in Frage.

Der Einsatz von wiederverwendbaren Steuerelementen wird möglich, weil einerseits alle Tragwerksplaner beim Skizzieren von statischen Systemen und Schnittgrößenverläufen bestimmten berufstypischen Konventionen folgen. Anderseits sind die üblichen Nachweise (z.B. für Biegung, Schub, Stabilität und Verformung) für die gängigen Bauweisen (Massivbau, Stahlbau, Holzbau) genormt. Infolgedessen ähneln sich Tragwerksberichte für vergleichbare Bauvorhaben sehr stark und unterscheiden sich nur durch die konkreten Zahlenwerte. Diese immer wiederkehrenden Bestandteile werden im Folgenden als Dokumentbausteine bezeichnet.

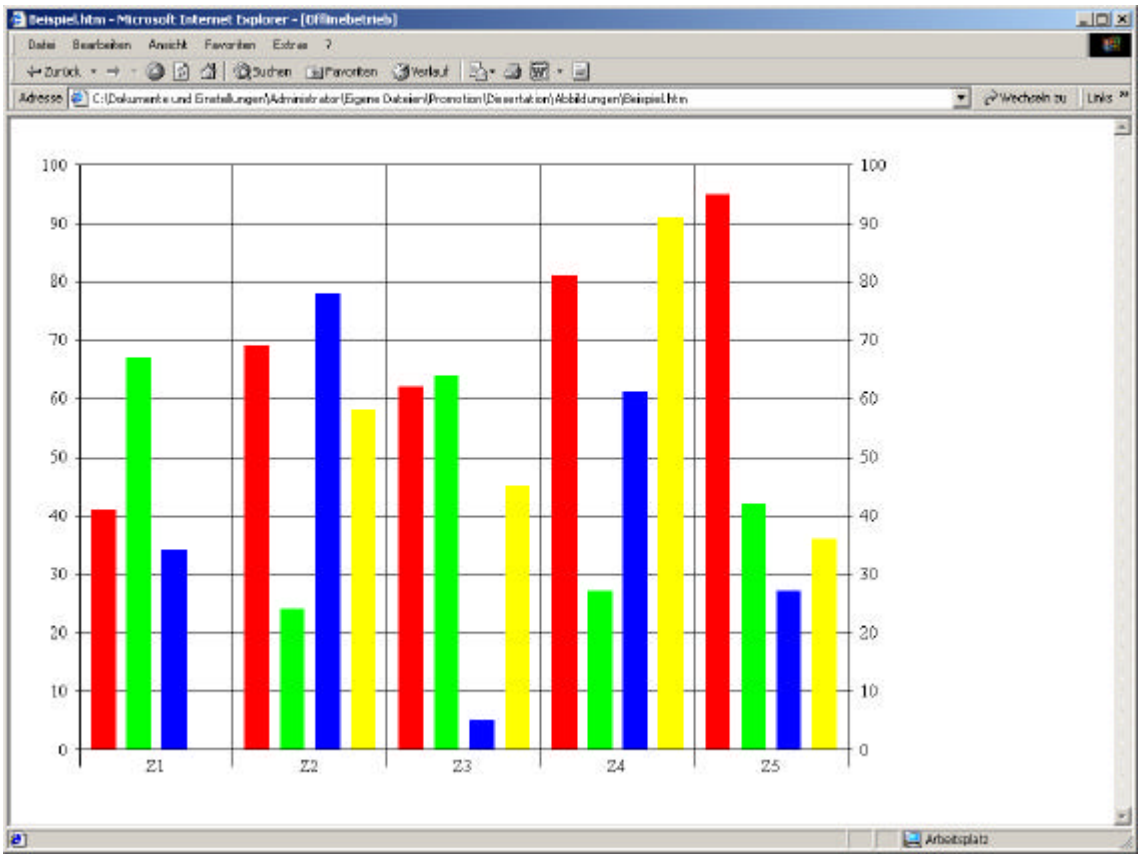


Abbildung 7.1: Beispiel für Container (Internet Explorer) und Steuerelement (Chart Control)

Alle Elemente der Nutzeroberfläche müssen über eine möglichst konsistente Darstellung verfügen. Beim gleichzeitigen Einsatz von Steuerelementen verschiedener Hersteller kann die geforderte konsistent gestaltete Nutzeroberfläche nicht ohne weiteres erreicht werden.

Die Steuerelemente müssen folglich über eine flexible Darstellung verfügen, so dass sie sich entsprechend der Vorgaben des Entwicklers beziehungsweise des Anwenders präsentieren können. Das bedeutet, dass unter anderem Schriftart und -größe, Zeilenabstand oder Tabulatoren durch externe Vorgaben beeinflusst werden können. Weiterhin müssen im Bereich der Tragwerksplanung ingenieurspezifische Vorgaben, beispielsweise bezüglich der verwendeten Einheiten oder der Anzahl signifikanter Ziffern berücksichtigt werden können.

7.1.3 Nutzung von Standard-Software

Neben den fachspezifischen Besonderheiten, welche die Entwicklung von spezialisierten Software-Lösungen erfordern, gibt es auch Funktionsmerkmale, die in standardisierter Form in fast allen Software-Produkten eingesetzt werden können. Als Konsequenz wird beispielsweise die für Textverarbeitung, Tabellenkalkulation, Datenbanken, CAD oder Kommunikation benötigte Funktionalität heute nahezu vollständig durch Standard-Software bereitgestellt. Die durch die vielfältigen Anwendungsmöglichkeiten ermöglichte Massenfertigung erlaubt gleichermaßen niedrige spezifische Entwicklungskosten, hohe Leistungsfähigkeit und Zuverlässigkeit sowie software-technische Aktualität.

Aus diesen Gründen ist es für spezialisierte Software-Lösungen im Allgemeinen nicht zweckmäßig, Standardfunktionalität selbst zu implementieren. In den meisten Fällen stehen weder das benötigte Know-how noch entsprechende Ressourcen für Forschung, Entwicklung, Qualitätssicherung und Wartung zur Verfügung. Somit besteht zur möglichst weitgehenden Nutzung von Standard-Software keine Alternative. Außerdem erfordert auch die Realisierung von allgemeinen, nicht-fachspezifischen Merkmalen wie Visualisierung in einer Layout-Ansicht, Druckvorschau, Druckfunktionalität, Funktionalität zur Unterstützung von OLE-Objekten und Steuerelementen einen sehr hohen Entwicklungsaufwand. Standard-Software kann auch für diese Aufgaben gleichermaßen leistungsfähige und kostengünstige Implementierungen zur Verfügung stellen.

Die Einbindung von Standard-Software in spezialisierte Lösungen erweist sich jedoch als sehr problematisch. Zwar verfügt mittlerweile der überwiegende Teil der Standard-Software über leistungsfähige Programmierschnittstellen, die eine Erweiterung erlauben. Damit kann Standard-Software ohne Schwierigkeiten um zusätzliche Aspekte ergänzt werden. Soll jedoch spezialisierte Software gezielt Funktionalität einbinden, die durch Standard-Software bereitgestellt wird, müssten die Nutzeroberflächen der beiden Anwendungen zusammengeführt werden. Derartige weitgehende Modifikationen können mit den gängigen Programmierschnittstellen allein nicht realisiert werden.

Die derzeit einzig gangbare Möglichkeit stellt die Einbindung eines Dokuments einer Standard-Software als Container-Dokument für spezielle Erweiterungen dar (Abbildung 7.1). Während das Container-Dokument die Funktionsmerkmale der Standard-Software zur Verfügung stellt, können spezialisierte Steuerelemente in das Container-Dokument eingebunden werden. Für die Tragwerksplanung bieten sich insbesondere Container-Dokumente von Internet-Browsern, Textverarbeitungen, Tabellenkalkulationen, Mathematik- und CAD-Software an.

Der Internet-Browser hat sich als Standard-Container für Steuerelemente etabliert. Folglich ist es nahe liegend, einen Internet-Browser als Standard-Software in eine Anwendung für die Tragwerksplanung einzubinden. Allerdings sind Internet-Browser für das passive Betrachten von WWW-Dokumenten optimiert. Der Tragwerksplaner benötigt jedoch ein leistungsfähiges Werkzeug zum aktiven Erstellen und Modifizieren von Dokumenten. Ein Internet-Browser erscheint deshalb als Container-Anwendung für die Tragwerksplanung nicht geeignet.

Üblicherweise benutzt der Tragwerksplaner eine Textverarbeitung zur Erstellung des Tragwerksberichts, weshalb die Verwendung von Dokumenten einer Textverarbeitung als Container besonders zweckmäßig erscheint. Damit können sämtliche Funktionsmerkmale von Textverarbeitungen wie Layout-Ansicht mit Seitenumbrüchen, Kopf- und Fußzeilen, Formatvorlagen, Gliederungen, Inhaltsverzeichnissen, Hyperlinks, Rechtschreibprüfung und so weiter mit sehr geringem Entwicklungsaufwand zur Verfügung gestellt werden. Nachteilig ist jedoch, dass sich Berechnungen üblicherweise nicht parametrisieren lassen.

Speziell mit Hinblick auf die gewünschte Parametrisierbarkeit stellen Dokumente von Tabellenkalkulationen eine weitere Option dar. Der Vorteil von parametrisierbaren Berechnungen muss aber durch ein sehr unflexibles Layout erkauft werden. Text und Grafiken müssen in ein festes Tabellenraster eingepasst werden. Die gegenwärtig zur Verfügung gestellte Textverarbeitungsfunktionalität ist für die Arbeit des Tragwerksplaners nicht ausreichend. Beispielsweise wird weder Fließtext angeboten noch können Inhaltsverzeichnisse automatisch erstellt werden. Die Parametrisierbarkeit ist zwar praktisch, wegen der Referenzierung der Zellen mittels Zeilen- und Spaltenangaben aber sehr unübersichtlich. Darüber hinaus kann sich der Anwender entweder die Werte oder die dahinterliegenden Formeln anzeigen lassen. Die Nachvollziehbarkeit der Tragwerksanalyse wird dadurch in einem nicht akzeptablen Maß eingeschränkt.

Die Nachvollziehbarkeit von Berechnungen ist bei Mathematik-Software wesentlich besser als bei Tabellenkalkulationen, zumindest wenn eine Layout-Ansicht möglich ist. Nachteilig ist jedoch, dass alle Variablen in der Regel dokumentweit gültig sind, was bei komplexen Rechenblättern eine gravierende Einschränkung darstellt. Die angebotene Textverarbeitungsfunktionalität ist wie bei den Tabellenkalkulationen nicht ausreichend.

Die Verwendung eines CAD-Systems als Containeranwendung stellt keine sinnvolle Option für die Tragwerksplanung dar. Moderne CAD-Systeme arbeiten modellorientiert und sind deshalb nicht für eine dokumentzentrierte Verwaltung des Tragwerksberichtes geeignet.

Zusammenfassend wird festgestellt, dass nach dem gegenwärtigen Stand der Technik Dokumente von Standard-Textverarbeitungen am besten als Container für fachspezifische Erweiterungen geeignet sind. Da die durch sie bereitgestellte Funktionalität ohnehin für die Erstellung des Tragwerksberichtes benötigt wird, kann eine deutliche Verringerung des Entwicklungsaufwandes erreicht werden.

7.2 Unabhängige Erweiterbarkeit von Software-Systemen

Aufgrund des enormen Entwicklungsumfanges kann eine vollständige Palette komplexer Anwendungen, wie sie der praktisch tätige Tragwerksplaner für seine Arbeit benötigt, heute nicht mehr von einem einzigen Anbieter entwickelt werden. Da die bestehenden Lösungen nicht alle Wünsche der Anwender abdecken, werden immer wieder Anwendungen entwickelt, die sich von den vorhandenen nur durch wenige Details unterscheiden. Um derartige kostspielige Mehrfachentwicklungen unnötig zu machen, wäre es notwendig, dass unabhängige Software-Entwickler eine bestehende Lösung um die aus ihrer Sicht fehlenden Aspekte erweitern können.

Infolgedessen wird eine Erweiterbarkeit von Software-Systemen durch unabhängige Software-Entwickler als mindestens ebenso wichtig erachtet wie die Wiederverwendung von Software-Bausteinen. Durch die Möglichkeit der Erweiterung können existierende Lösungen an individuelle Anforderungen angepasst werden. Die Entwicklung kann auf Vorarbeiten aufbauen, Entwicklungszeit und -kosten können drastisch gesenkt werden. Gleichzeitig wird eine Zersplitterung des Marktes in eine Vielzahl inkompatibler Anwendungen vermieden.

Eines der erfolgreichsten Beispiele für ein im Bauwesen relevantes erweiterbares System stellt die von der Firma Autodesk entwickelte CAD-Standard-Software AutoCAD dar. Während die gesamte Basisfunktionalität (horizontale Funktionalität) für CAD-Systeme von Autodesk entwickelt wird, können unabhängige Software-Firmen über die objektorientierte AutoCAD Runtime Extension (Object ARX) fachspezifische Erweiterungen (vertikale Funktionalität) hinzufügen (Abbildung 7.2). Nachteilig ist jedoch, dass das Object ARX-Konzept auf einer proprietären Technologie basiert und nicht auf einem der etablierten Standards für Komponentenmodelle aufbaut.

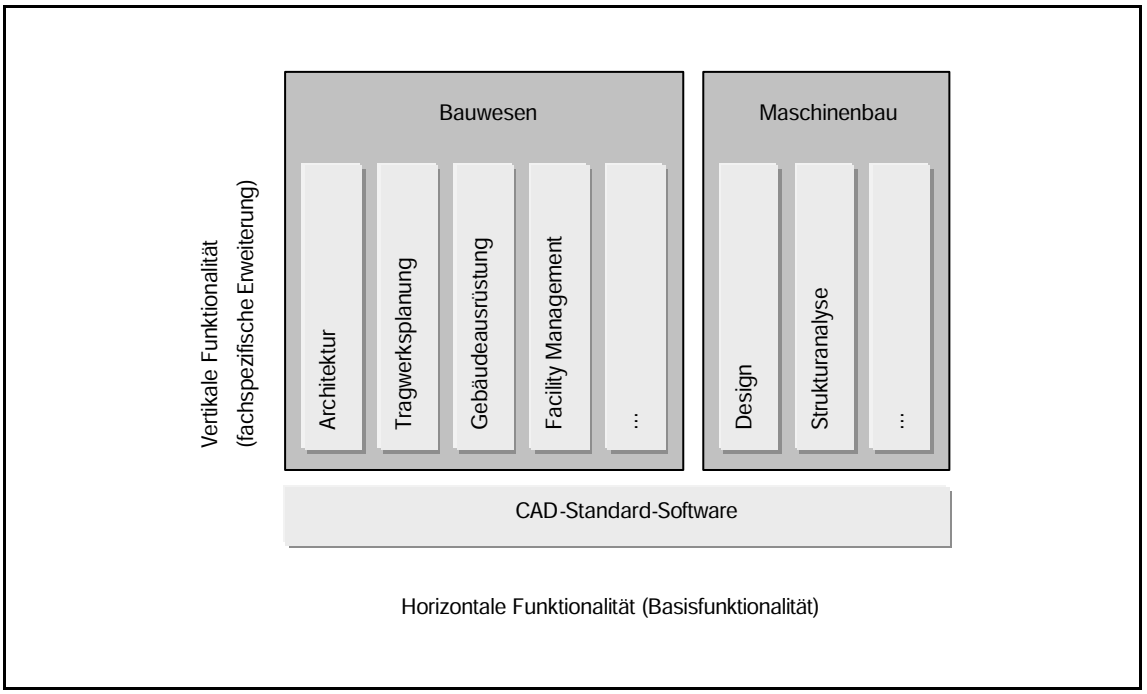


Abbildung 7.2: Horizontale und vertikale Funktionalität am Beispiel von CAD-Software

Mit diesem Ansatz kann durch die komplette Wiederverwendung eines kommerziell verfügbaren Software-Produktes zu äußerst wettbewerbsfähigen Konditionen ein spezialisiertes Produkt entwickelt werden. Die Kosten für Wartung und Weiterentwicklung beschränken sich auf die vertikale Funktionalität. Von der Weiterentwicklung des Basisproduktes profitiert automatisch auch die spezialisierte Anwendung. Die Nutzeroberfläche der fachspezifischen Erweiterung folgt üblicherweise den Konventionen der Standard-Software, wobei große Teile sogar völlig identisch sind. Dadurch kann die Einarbeitungszeit für schon mit dem Basisprodukt oder anderen vertikalen Erweiterungen vertraute Anwender deutlich reduziert werden.

Eine Besonderheit eines erweiterbaren CAD-Systems ist, dass Erweiterungen zwar durch verschiedene unabhängige Software-Entwickler vorgenommen werden können, dass im Regelfall jedoch keine Kombination der unabhängig voneinander entwickelten Erweiterungen stattfindet. Dies ist weder notwendig noch sinnvoll, da für eine fachspezifische Aufgabe nur eine überschaubare Zahl von zusätzlichen Objektklassen (im Bauwesen beispielsweise Wände, Türen, Fenster, Decken, Stützen, Treppen, Dächer) benötigt wird. Die Implementierungen aller benötigten Objektklassen werden im Regelfall durch die fachspezifische Erweiterung eines einzigen Software-Herstellers bereitgestellt.

Um den Wiederverwendungsgrad zu steigern, können für diese Objektklassen Standard-Implementierungen angeboten werden. Neben der dadurch ermöglichten kostengünstigen Wiederverwendung von Funktionalität ergeben sich auch neue Möglichkeiten zur herstellerübergreifenden Integration, da die grundsätzliche Datenstruktur aller fachspezifischen Erweiterungen identisch ist. Somit ist eine umfassende Integration der Fachapplikationen möglich, ohne dass dafür die Festlegung branchenweiter Standards abgewartet werden muss (Bittrich, 1998). Beispielsweise forciert die Firma Autodesk aus diesen Überlegungen heraus die Entwicklung des AEC-spezifischen Object Modeling Frameworks (OMF) einschließlich der auf dieser Standard-Funktionalität basierenden bauwesenspezifischen Erweiterung Architectural Desktop (ADT).

Für bestimmte Aufgabenstellungen ist es notwendig, dass auch unabhängig voneinander entwickelte Erweiterungen miteinander kombiniert werden können. Die Interoperabilität der parallel entwickelten Module muss möglich sein, ohne dass dazu eine Integration auf der Ebene des Quell-Codes erforderlich ist. Zur Unterscheidung von herkömmlichen erweiterbaren Systemen werden derartige Systeme als unabhängig erweiterbare Systeme bezeichnet. Eine Möglichkeit zur Realisierung eines unabhängig erweiterbaren Systems stellen die in Kapitel 5.3.2 diskutierten Component Frameworks dar.

Ein Component Framework erlaubt die unabhängige Erweiterung eines Systems in vordefinierte Dimensionen. Mit Blick auf die bauteilorientierte Tragwerksplanung bedeutet dies, dass es jederzeit möglich sein muss, ein System durch das Hinzufügen von Software-Komponenten um Funktionalität zur Bearbeitung neuer Bauteiltypen (zum Beispiel Platten, Unterzüge, Stützen, Fundamente) zu erweitern. Für viele dieser Bauteiltypen sind weitere Klassifizierungen möglich (zum Beispiel Einfeld- und Mehrfeldsysteme), wodurch sich eine sehr große Zahl von möglichen spezialisierten Anwendungen ergibt, die sich häufig nur in wenigen Details unterscheiden.

Um unnötige Mehrfachentwicklungen zu vermeiden, ist es zweckmäßig, dass bei der Implementierung der Bauteiltypen – wie in Kapitel 7.1.1 und Kapitel 7.1.2 beschrieben – sowohl auf Software-Komponenten ohne Nutzeroberfläche (z.B. für die Tragwerksanalyse) als auch auf Software-Komponenten mit eigener Nutzeroberfläche (z.B. für Dokumentbausteine) zurückgegriffen wird. Die Funktionalität zur Bearbeitung eines Bauteiltyps bildet dann eine grobgranulare Software-Komponente, die intern auf feingranulare Software-Komponenten zurückgreift. Neben der Möglichkeit der direkten Erweiterung in der Dimension der Bauteiltypen kann das System auch indirekt in der Dimension der Dokumentbausteine erweitert werden (Abbildung 7.3).

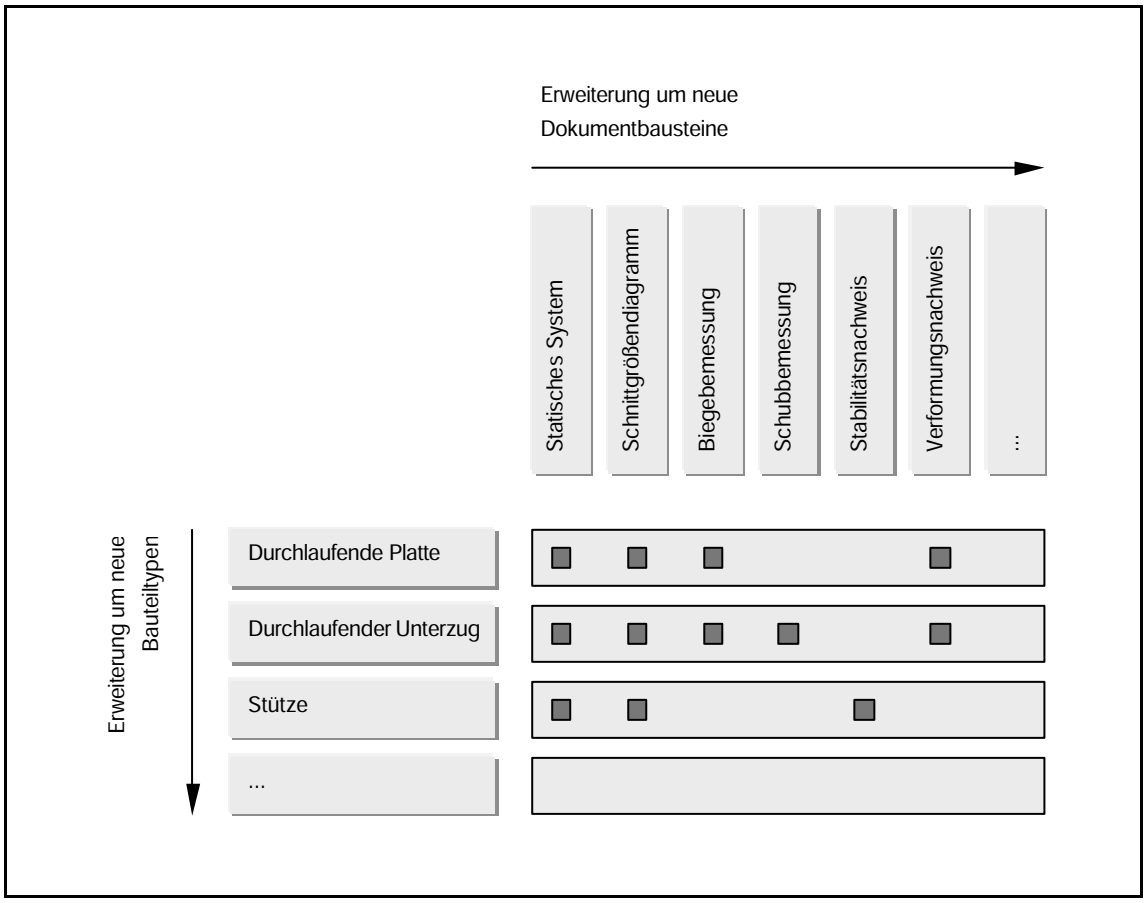


Abbildung 7.3: Component Framework für die Tragwerksplanung

Das Prinzip der unabhängigen Erweiterbarkeit hat weitreichende Auswirkungen auf die Wahl des einzusetzenden Integrationskonzeptes. Offensichtlich können die bekannten Probleme der Entwicklung eines einheitlichen Datenmodells für die Tragwerksplanung nicht allein durch den Übergang von Dateischnittstellen auf Software-Schnittstellen gelöst werden. Das eigentliche Problem liegt aber darin, dass einem unabhängig erweiterbaren System jederzeit neue Bestandteile hinzugefügt werden. Damit ist auch eine Erweiterung um Bestandteile möglich, die in der Entwurfsphase nicht berücksichtigt worden sind. Bezogen auf ein unabhängig erweiterbares System für die Tragwerksplanung bedeutet dies, dass das System zu keinem Zeitpunkt als Ganzes analysiert werden kann und somit aus den analysierten Nutzungsszenarien kein allgemein gültiges und vollständiges Tragwerksmodell abgeleitet werden kann.

Da ein standardisiertes Produktdatenmodell für die Tragwerksplanung ebenfalls noch nicht definiert werden konnte, kann eine Integration nur durch ein Integrationskonzept ohne Interpretation der Semantik erfolgen, welches stattdessen das Fachwissen des Anwenders nutzt. Dies stellt keine gravierende Einschränkung dar, da in Kapitel 6.3.4 ohnehin ein derartiges Tragwerksmodell als ingenieurgemäße Lösung identifiziert wurde.

7.3 Verbunddokumente als Nutzeroberfläche

Eine besondere Schwierigkeit bei der Realisierung von unabhängig erweiterbaren Systemen stellt das Design der Nutzeroberfläche dar. Infolge des erweiterbaren Funktionsumfangs muss offensichtlich die Nutzeroberfläche ebenfalls erweiterbar sein. Verbunddokumente stellen dafür einen vielversprechenden Ansatz dar.

Die zusätzliche Funktionalität der in die Verbunddokumente integrierten Software-Komponenten mit eigener Nutzeroberfläche muss durch geeignete Dialogelemente erschlossen werden. Üblicherweise werden zu diesem Zweck Dialogboxen benutzt. Der Aufruf der Dialogboxen erfolgt konventionsgemäß über Kontextmenüs oder Doppelklick. Alternativ kann der Aufruf auch über zusätzliche, vom Steuerelement bereitgestellte Menüs oder Symbolleisten erfolgen. Die Integration von Steuerelementen in Verbunddokumente erfordert somit nur sehr geringe Änderungen an der Nutzeroberfläche. Die benötigten Anpassungen werden durch die in Kapitel 4.2 vorgestellten Standards für Verbunddokumente ermöglicht.

Das ursprüngliche Konzept für Verbunddokumente basierte auf einer vollständigen Trennung von Container und Steuerelementen. Mittlerweile wurde erkannt, dass auch eine weniger strikte Auslegung vorteilhaft sein kann (Weck, 1996). Beispielsweise kann ein Steuerelement prüfen, in welchem Container es sich befindet. Sofern der Container zu den vom Steuerelement unterstützten Containern gehört, kann es seine Umgebungseigenschaften abfragen. Diese Informationen können benutzt werden, um eine angepasste und einheitliche Darstellung der Steuerelemente zu erreichen. So kann ein Steuerelement in einem Textverarbeitungsdokument automatisch Schriftart und -größe, Zeilen- und Absatzabstände sowie Tabulatoren übernehmen. Wird hingegen das Steuerelement in einen anderen Container eingefügt, sind diese Funktionsmerkmale nicht verfügbar.

Im Fall der Tragwerksplanungs-Software erscheint es sinnvoll, eine einheitliche Rahmenanwendung zu erstellen, die zur Verwaltung aller zu einem Projekt gehörenden Positionen dient. In diese Rahmenanwendung werden Textverarbeitungsdokumente eingebunden. Entsprechend dem momentan bearbeiteten Bauteiltyp wird die benötigte Funktionalität durch die im Textverarbeitungsdokument enthaltenen Steuerelemente bereitgestellt. Die notwendigen Dialogelemente werden durch die Steuerelemente selbst verfügbar gemacht. Dadurch ist eine präzise Zuschneidung der Nutzereroberfläche auf ein spezielles Problem möglich. Als Nebeneffekt kann dadurch die Komplexität der Nutzer-Software-Interaktion reduziert werden, womit auch eine deutliche Verringerung des Entwicklungsaufwandes erreicht werden kann.

Die Benutzung von Textverarbeitungsdokumenten als Nutzeroberfläche erlaubt es, das gleiche Bedienkonzept für die Funktionalität zur Textverarbeitung und für die Funktionalität zur Tragwerksanalyse zu verwenden. Insofern stellt die Möglichkeit der direkten Repräsentation und Manipulation des Verbunddokuments für den Anwender eine wesentliche Vereinfachung dar.

7.4 Verbunddokumente als Datenspeicher

Herkömmliche Applikationen basieren auf einem einheitlichen Datenmodell, ihre Datenhaltung erfolgt in Dateien mit homogenem Inhalt (beispielsweise nur Text oder Pixelgrafik). Auf Verbunddokumenten basierende Applikationen müssen nicht nur die Daten des eigenen Datenmodells speichern können, sondern auch die Daten beliebiger, voneinander völlig unabhängiger Objekte.

Zu diesem Zweck unterstützen die in Kapitel 4.2 untersuchten Standards für Verbunddokumente sogenannte Verbunddateien. Eine Verbunddatei erlaubt die Definition einer Struktur von Ordnern und Daten innerhalb einer Datei (Abbildung 7.4). Diese interne Struktur ist für den Anwender nicht transparent. Damit können die Daten für jedes in einem Verbunddokument enthaltene Steuerelement in einem separaten Ordner abgelegt werden. Die Datenhaltung der einzelnen Steuerelemente ist somit völlig unabhängig voneinander. Trotzdem werden die fragmentierten Daten der Steuerelemente aus der Perspektive des Anwenders zu einer übersichtlichen Einheit zusammengefasst.

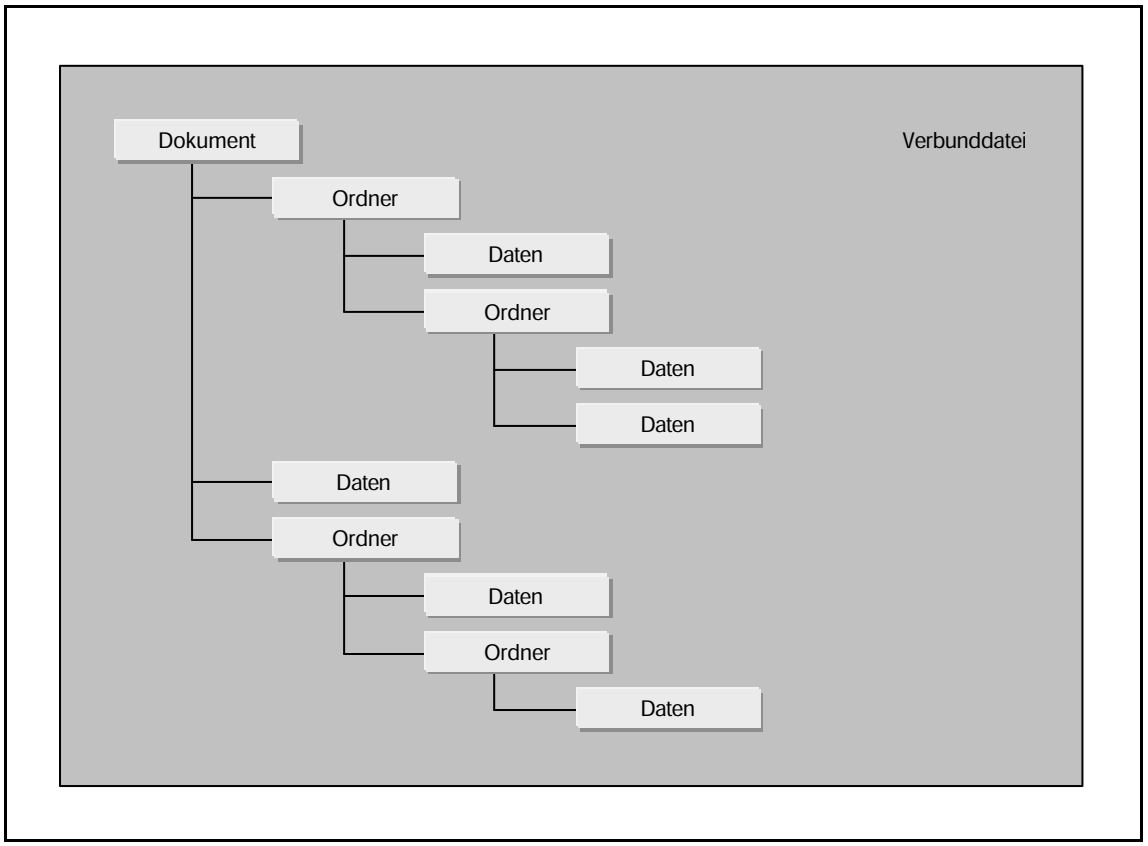


Abbildung 7.4: Exemplarischer Aufbau einer Verbunddatei

Die Nutzung von Verbunddateien bietet auch software-technisch große Vorteile. Bei der Entwicklung der Steuerelemente muss nicht mehr jeweils ein eigenes Datenmodell definiert werden. Der Entwickler legt nur noch fest, welche Informationen persistent gemacht werden müssen. Zum Speichern und Laden wird vom Container bereitgestellte Infrastruktur benutzt. Der Zeitpunkt und die Durchführung von Speicher- und Ladevorgängen wird ebenfalls vom Container bestimmt.

Kapitel 8
Entwurf der Architektur eines Tragwerkseditors

In diesem Kapitel wird unter Berücksichtigung der zuvor erläuterten Anforderungen der Tragwerksplaner und der Software-Entwickler der Entwurf der Architektur eines Tragwerkseditors dokumentiert. Zu Beginn werden die allgemeinen Anforderungen an die Architektur des Tragwerkseditors nochmals zusammengefasst. Es folgt eine Begründung der Wahl der Entwicklungsplattform. Danach werden Vorteile und Nachteile der denkbaren Varianten zur Realisierung von Tragwerkseditor, Dokumentbausteinen, Bauteiltypen und der sonstigen Software-Komponenten diskutiert.

8.1 Vorbemerkungen

Vor Beginn des Entwurfs der Architektur des Tragwerkseditors werden unter Bezugnahme auf die Erkenntnisse der vorangegangenen Kapitel die Anforderungen an die ideale Systemarchitektur beschrieben.

Aus der Sicht der Tragwerksplaner soll der Tragwerkseditor wie in Kapitel 6 erläutert über spezialisierte und dokumentzentrierte Nutzeroberflächen verfügen. Soweit wie möglich sind die gewohnten Nutzereroberflächen von Standard-Software zur Textverarbeitung einzubinden, die durch Dokumentbausteine an die fachspezifischen Erfordernisse der Tragwerksplaner angepasst werden. Damit die Tragwerksplaner stärker in Entwicklung und Anpassung von Tragwerksplanungs-Software einbezogen werden können, muss die Architektur des Tragwerkseditors eine Trennung der Aufgaben von professionellen Software-Entwicklern und Tragwerksplanern unterstützen.

Aus software-technischer Sicht soll die Architektur eine maximale Wiederverwendung bereits implementierter Funktionalität ermöglichen. Dazu ist ein möglichst hoher Codeanteil in Form von Software-Komponenten ohne und mit eigener Nutzeroberfläche zu implementieren. Bei der Implementierung der benötigten Funktionalität ist soweit wie möglich auf entsprechende Standard-Software zurückzugreifen. Weiterhin sollte die Architektur des Tragwerkseditors auch eine Erweiterung durch unabhängige Software-Entwickler erlauben. Wie in Kapitel 7 beschrieben, soll die Implementierung des Tragwerkseditors auf den Technologien für Verbunddokumente und Component Frameworks basieren.

Infolge der Vorgabe nach maximaler Wiederverwendung vorhandener Implementierungen kann der Entwurf der Architektur nicht auf dem klassischen Weg als Top-Down-Approach erfolgen, bei dem zunächst eine grobe Konzeption festgelegt wird, die anschließend verfeinert und im Detail ausgearbeitet wird. Stattdessen müssen mit Hilfe eines Bottom-Up-Approach die infrage kommenden Software-Komponenten identifiziert und evaluiert werden. Ein Großteil der Design-Entscheidungen wird folglich bereits mit der Auswahl der zu benutzenden Software-Komponenten getroffen.

8.2 Festlegung von Plattform und Komponentenmodell

Bereits zu Beginn der Entwurfsphase muss eine Entscheidung getroffen werden, auf welcher Plattform das Software-Projekt realisiert werden soll beziehungsweise ob eine plattformunabhängige Lösung angestrebt wird.

Die Realisierung einer plattformunabhängigen Lösung erfordert stets einen höheren Aufwand als die einer plattformspezifischen Lösung. Dies resultiert zum einen aus dem höheren Evaluierungsaufwand für die verschiedenen Plattformen. Zum anderen existieren wesentliche Unterschiede, welche Basisdienste vom jeweiligen Betriebssystem zur Verfügung gestellt werden. Um den Entwicklungsaufwand zu beschränken, wird in vielen Fällen auf die Implementierung von Funktionsmerkmalen verzichtet, die nicht von allen anvisierten Plattformen unterstützt werden. Kann auf eine bestimmte Funktionalität nicht verzichtet werden, muss diese durch eigene Entwicklungen bereitgestellt werden. Sofern das überhaupt möglich ist, resultiert daraus zusätzlicher Entwicklungsaufwand. Weitere Probleme können sich aus den je nach Plattform unterschiedlichen Gestaltungsrichtlinien für Nutzeroberflächen ergeben. Aus den genannten Gründen sind plattformunabhängige Lösungen immer Kompromisslösungen, die den kleinsten gemeinsamen Nenner der beteiligten Plattformen repräsentieren. In vielen Fällen stellt die plattformspezifische Lösung die bessere Wahl dar. Plattformunabhängige Lösungen sollten nur dann zum Einsatz kommen, wenn dies aufgrund inhomogener Zielsysteme erforderlich ist.

Für das Gebiet der Tragwerksplanung wird plattformunabhängige Software nur in einigen Sonderfällen (beispielsweise zur Einbindung von Hochleistungsrechnern oder für Internet-basierte Anwendungen) benötigt. Aufgrund des hohen Verbreitungsgrades der Windows-Plattform und dem damit einhergehenden umfassenden Angebot an Standard-Software kann diese als alleinige Zielplattform gewählt werden. Das eng mit der Windows-Plattform verknüpfte Komponentenmodell COM stellt sich auch im direkten Vergleich mit den in Kapitel 4 diskutierten alternativen Komponentenmodellen Java und CORBA als innovativ und leistungsfähig dar. Infolge der langen Verfügbarkeit ist COM ausgereift und bietet mit OLE und ActiveX den gegenwärtig leistungsfähigsten Standard für Verbunddokumente. Für die Software-Entwicklung steht mit dem Microsoft Visual Studio eine leistungsfähige Entwicklungsumgebung zur Verfügung.

8.3 Entwurf des Tragwerkseditors

Den Kern der entworfenen Architektur bildet der Tragwerkseditor. Er dient zum einen der Erstellung des Tragwerksberichtes und muss folglich über alle Merkmale einer modernen Textverarbeitung verfügen. Gemäß den in Kapitel 6 erläuterten Konzepten soll ein Projekt den Teiltragwerken entsprechend in Positionen strukturiert werden. Somit wird außerdem Funktionalität zur Verwaltung aller zu einem Projekt gehörenden Positionen benötigt. Folglich muss der Tragwerkseditor die Funktionalität zur Verwaltung von Projekten mit einer speziell auf die Bedürfnisse des Tragwerksplaners zugeschnittenen Textverarbeitung kombinieren.

Die gängige Standard-Software zur Textverarbeitung erlaubt jedoch keine Verwaltung von Projekten. Da die übergeordnete Instanz eines Projektes nicht von Beginn an vorgesehen wurde, ist auch eine nachträgliche Erweiterung von Standardtextverarbeitungs-Software um diesen Aspekt nicht möglich.

Aus diesen Gründen wird eine Eigenentwicklung des Tragwerkseditors als notwendig erachtet. Neben der Projektverwaltung muss dabei die Einbindung von Dokumenten einer Standard-Textverarbeitung als MDI-Child-Windows möglich sein. Prinzipiell bieten sich für diese Aufgabenstellung mit OLE Objects und Active Documents zwei Technologien an, die im Folgenden auf ihre Eignung hin untersucht werden.

8.3.1 Variante basierend auf OLE Objects

OLE Objects sind Objekte, die in Container-Dokumente anderer Anwendungen eingebettet werden können. Dabei können Menüs und Symbolleisten der Containeranwendung durch Elemente der Nutzeroberfläche der dem OLE Object zugeordneten fremden Anwendung ersetzt werden.

Die Entwicklung von OLE Servern und OLE Containern wird von allen relevanten Entwicklungswerkzeugen für die Windows-Plattform unterstützt. Da die Implementierung dieser OLE-Basisfunktionalität ausdrücklich Bestandteil der Richtlinien zur Gestaltung von Nutzeroberflächen ist (Microsoft, 1995b), wird entsprechende Funktionalität auch von nahezu allen Anwendungen angeboten.

Zur Bearbeitung eines OLE Objects muss dieses zunächst durch Doppelklick aktiviert werden. Anschließend wird das OLE Object innerhalb eines schraffierten Rahmens dargestellt und kann über die modifizierten Menüs und Symbolleisten bearbeitet werden (Abbildung 8.1). Wesentlich ist, dass OLE Objects keinerlei Kontrolle über die Containeranwendung haben, in der sie eingebettet sind. Die Größe der einzufügenden OLE Objects sollte die Seitengröße des Container-Dokuments nicht überschreiten. Andernfalls können zwar behelfsweise Bildlaufleisten eingeblendet werden, um die verdeckten Informationen sichtbar zu machen. Die Ausgabe auf einem Drucker ist jedoch stets auf den sichtbaren Teil beschränkt. Aufgrund dieser Unzulänglichkeiten stellen OLE Objects keinen ausreichend leistungsfähigen Ansatz zur Einbindung von Funktionalität von Standard-Software in den Tragwerkseditor dar.

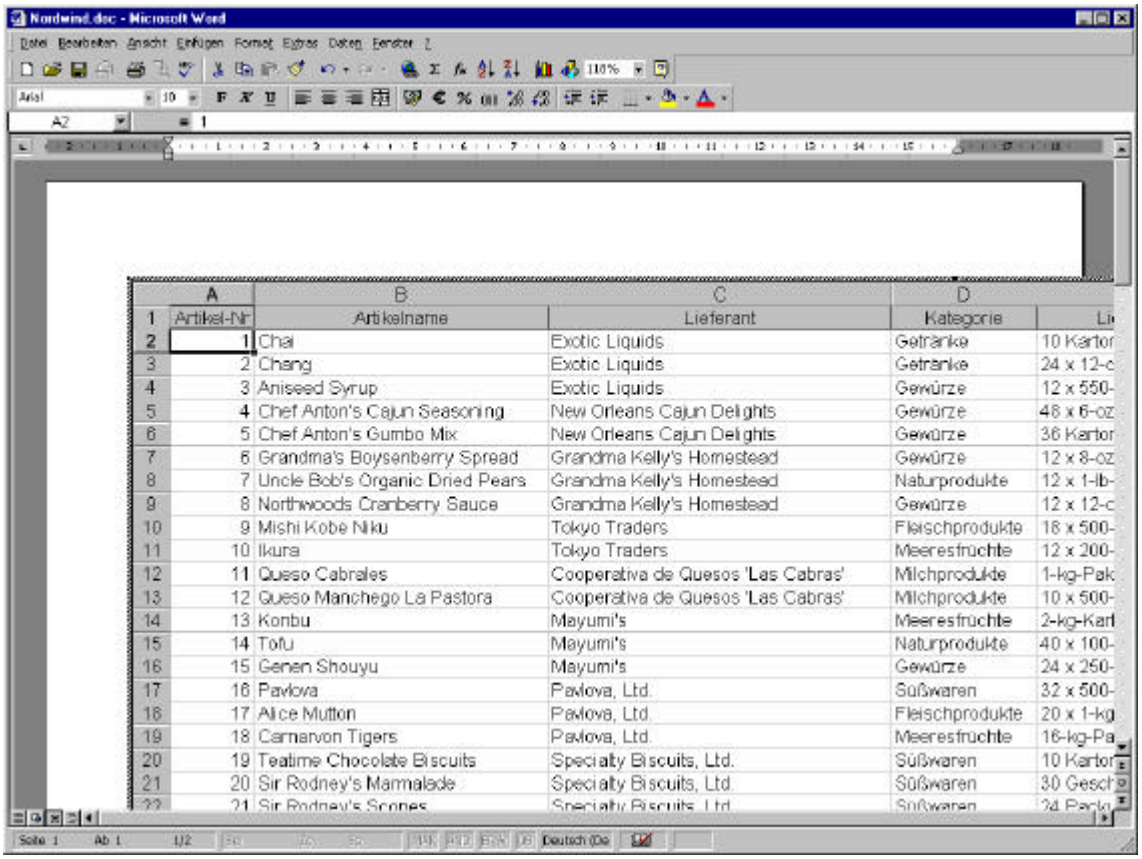
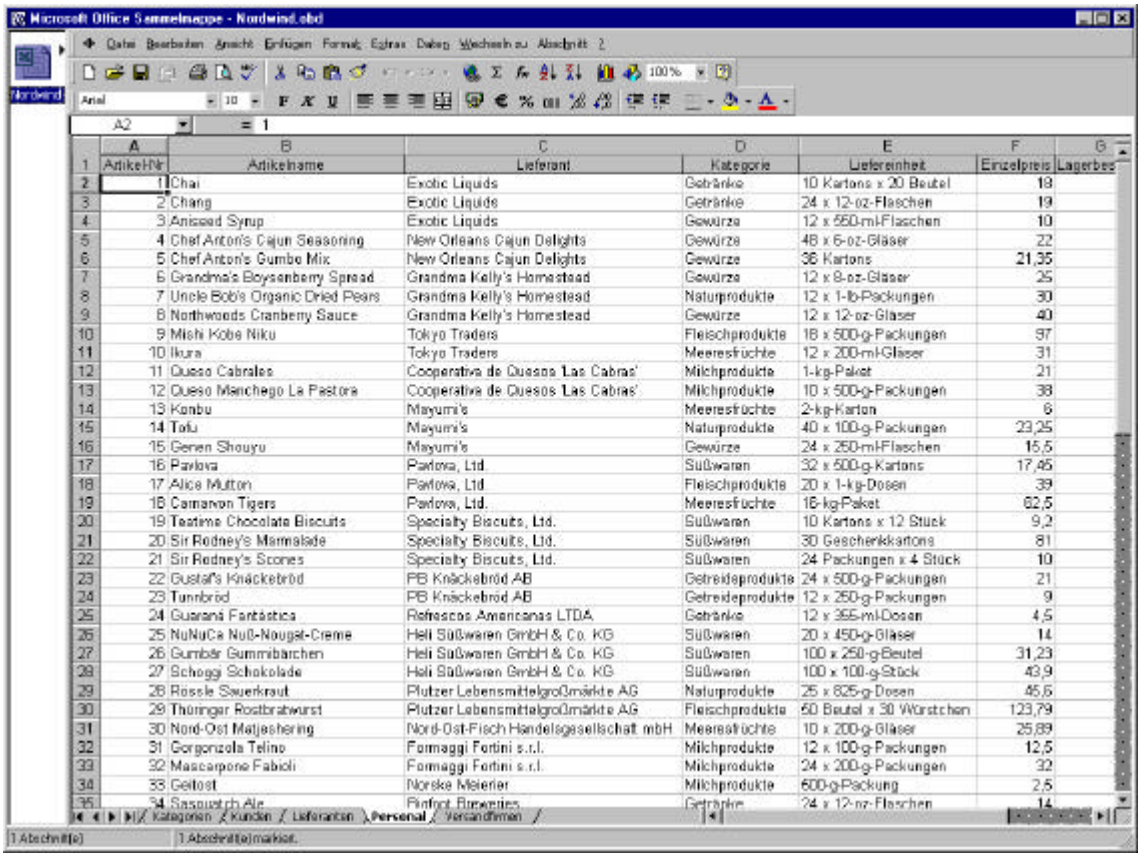


Abbildung 8.1: Beispiel Word-Dokument mit als OLE Object eingebetteter und aktivierter Excel-Tabelle

8.3.2 Variante basierend auf Active Documents

Da OLE Objects zur Bearbeitung vollständiger Dokumente nicht geeignet sind, wurde dieser Ansatz um das Konzept der sogenannten Active Documents erweitert. Active Documents sind komplette Dokumente, die einschließlich weiterer Elemente der Nutzeroberfläche (Menüs, Symbolleisten, Bildlaufleisten) in eine generische Container-Anwendung eingebettet werden können. Dazu wurden, wie in Abbildung 8.3 und Abbildung 8.4 dargestellt, neben den Interfaces für OLE Embedding Server, OLE Embedding Container und OLE In-Place Activation zusätzliche Interfaces für Active Document Server und Active Document Container definiert.

Ein Active Document Container kann alle der Spezifikation entsprechenden Active Documents aufnehmen. Active Documents haben die volle Kontrolle über die Container-Anwendung und somit auch über das Seiten-Layout. Active Documents sind ständig aktiv, die Darstellung erfolgt ohne Rahmen und entspricht im Wesentlichen dem Erscheinungsbild in der nativen Anwendung. Eine explizite Aktivierung wie bei den OLE Objects ist somit nicht erforderlich (Abbildung 8.2). Für den Nutzer ist eine auf der Active Document Technologie basierende Anwendung nicht von einer herkömmlichen Anwendung zu unterscheiden.



Artikel-Nr	Artikelname	Lieferant	Kategorie	Liefereinheit	Einzelpreis	Lagerbes
1	Chai	Exotic Liquids	Getränke	10 Kartons x 20 Beutel	18	
2	Chang	Exotic Liquids	Getränke	24 x 12-oz-Flaschen	19	
3	Aniseed Syrup	Exotic Liquids	Gewürze	12 x 550-ml-Flaschen	10	
4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	Gewürze	48 x 6-oz-Gläser	22	
5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Gewürze	36 Kartons	21,35	
6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead	Gewürze	12 x 8-oz-Gläser	25	
7	Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead	Naturprodukte	12 x 1-lb-Packungen	30	
8	Northwoods Cranberry Sauce	Grandma Kelly's Homestead	Gewürze	12 x 12-oz-Gläser	40	
9	Mishi Kobe Niku	Tokyo Traders	Fleischprodukte	18 x 500-g-Packungen	97	
10	Ikura	Tokyo Traders	Meeresfrüchte	12 x 200-ml-Gläser	31	
11	Queso Cabrales	Cooperativa de Quesos 'Las Cabras'	Milchprodukte	1-kg-Paket	21	
12	Queso Manchego La Pastora	Cooperativa de Quesos 'Las Cabras'	Milchprodukte	10 x 500-g-Packungen	38	
13	Konbu	Mayumi's	Meeresfrüchte	2-kg-Karton	6	
14	Tofu	Mayumi's	Naturprodukte	40 x 100-g-Packungen	23,25	
15	Genen Shouyu	Mayumi's	Gewürze	24 x 250-ml-Flaschen	15,5	
16	Pavlova	Pavlova, Ltd.	Süßwaren	32 x 500-g-Kartons	17,45	
17	Alice Mutton	Pavlova, Ltd.	Fleischprodukte	20 x 1-kg-Dosen	39	
18	Carnarvon Tigers	Pavlova, Ltd.	Meeresfrüchte	16-kg-Paket	62,5	
19	Teatime Chocolate Biscuits	Specialty Biscuits, Ltd.	Süßwaren	10 Kartons x 12 Stück	9,2	
20	Sir Rodney's Marmalade	Specialty Biscuits, Ltd.	Süßwaren	30 Geschenkkartons	81	
21	Sir Rodney's Scones	Specialty Biscuits, Ltd.	Süßwaren	24 Packungen x 4 Stück	10	
22	Gustaf's Knäckebröd	PB Knäckebröd AB	Getreideprodukte	24 x 500-g-Packungen	21	
23	Tunnbröd	PB Knäckebröd AB	Getreideprodukte	12 x 250-g-Packungen	9	
24	Guaraná Fantástica	Refrescos Americanas LTDA	Getränke	12 x 355-ml-Dosen	4,5	
25	NuNuCa Nuß-Nougat-Creme	Heli Süßwaren GmbH & Co. KG	Süßwaren	20 x 450-g-Gläser	14	
26	Gumbär Gummibärchen	Heli Süßwaren GmbH & Co. KG	Süßwaren	100 x 250-g-Beutel	31,23	
27	Schoggi Schokolade	Heli Süßwaren GmbH & Co. KG	Süßwaren	100 x 100-g-Stück	43,9	
28	Rössle Sauerkraut	Plutzer Lebensmittelgroßmärkte AG	Naturprodukte	25 x 825-g-Dosen	45,6	
29	Thüringer Rostbratwurst	Plutzer Lebensmittelgroßmärkte AG	Fleischprodukte	50 Beutel x 30 Würstchen	123,79	
30	Nord-Ost Matjeshering	Nord-Ost-Fisch Handelsgesellschaft mbH	Meeresfrüchte	10 x 200-g-Gläser	25,89	
31	Gorgonzola Telino	Formaggi Fortini s.r.l.	Milchprodukte	12 x 100-g-Packungen	12,5	
32	Mascarpone Fabioli	Formaggi Fortini s.r.l.	Milchprodukte	24 x 200-g-Packungen	32	
33	Geitost	Norske Meierier	Milchprodukte	500-g-Packung	2,5	
34	Sasquatch Ale	Bigfoot Breweries	Getränke	24 x 12-oz-Flaschen	14	

Abbildung 8.2: Beispiel Sammelmappe mit als Active Document aktivierter Excel-Mappe

Während nahezu alle Anwendungen über Funktionalität für OLE Objects verfügen, unterstützen nur wenige die Technologie der Active Documents. Beispiele für Anwendungen, die als Active Document Container benutzt werden können, stellen der Microsoft Internet Explorer, die Microsoft Sammelmappe und das Microsoft Visual Studio dar. Mit diesen Anwendungen können alle Dokumenttypen geöffnet werden, die der Spezifikation eines Active Document Servers entsprechen (Abbildung 8.3).

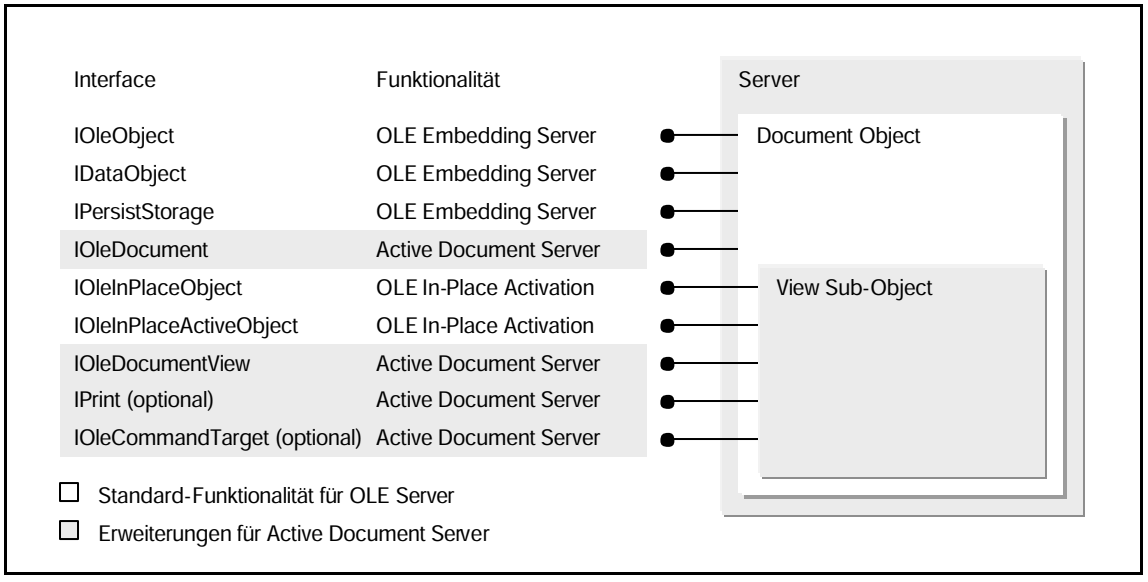


Abbildung 8.3: Interfaces eines Active Document Servers (Chappell, 1996)

Beispiele für Anwendungen, die als Active Document Server fungieren können, stellen der Microsoft Internet Explorer, Microsoft Word, Microsoft Excel, Microsoft PowerPoint und das Microsoft Visual Studio dar. Damit wäre es prinzipiell möglich, eine der genannten Anwendungen als Basis für einen Tragwerkseditor zu verwenden. Darin könnten Dokumente von Word als Active Documents integriert werden, da diese Anwendungen alle benötigten Interfaces implementieren (Abbildung 8.4). Diese Vorgehensweise wäre besonders attraktiv, da beispielsweise das Visual Studio die Organisation von Projekten unterstützt und somit den denkbar niedrigsten Entwicklungsaufwand erfordern würde. Da eine Anpassung der verfügbaren Active Document Container nur in sehr beschränktem Umfang möglich ist, wird eine Eigenentwicklung des Tragwerkseditors als notwendig erachtet.

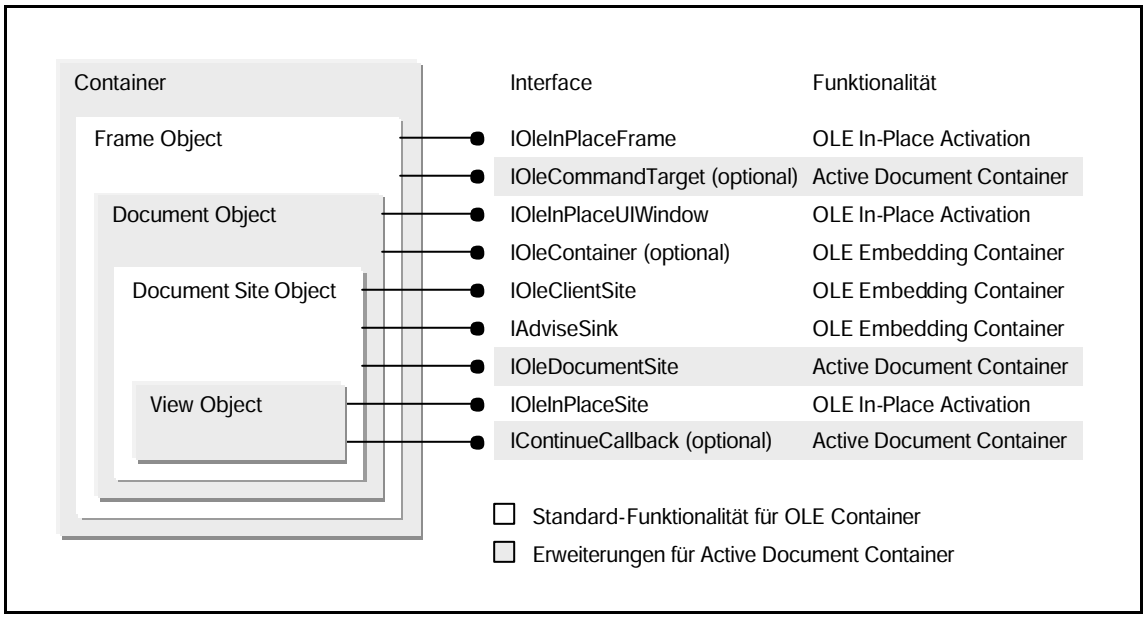


Abbildung 8.4: Interfaces eines Active Document Containers (Chappell, 1996)

8.4 Entwurf der Dokumentbausteine

Alle Dokumentteile des Tragwerksberichtes, die in ähnlicher Form immer wiederkehren, werden zu Dokumentbausteinen zusammengefasst (Abbildung 8.5). Damit wird ein wesentlicher Teil der fachspezifischen Funktionalität des Tragwerkseditors bereitgestellt. Im Regelfall dient jeder Dokumentbaustein zur Durchführung eines standardisierten Nachweises. In bestimmten Fällen – beispielsweise zur Ermittlung der Schnittgrößen – greifen Dokumentbausteine auch auf ausgegliederte Software-Komponenten ohne Nutzeroberfläche zurück. Mit Ausnahme der Dokumentbausteine zur Visualisierung von statischen Systemen und Schnittgrößendiagrammen wird für alle Dokumentbausteine die Tabellenform als zweckmäßig erachtet. Per Voreinstellung sind 4 Spalten für Bezeichnung, Wert, Einheit und Kommentar vorgesehen. Um eine gute Lesbarkeit der Dokumente sicherzustellen, werden umfassende Formatierungsmöglichkeiten als zwingend erforderlich erachtet. Das heißt, dass die Verwendung unterschiedlicher Fonts (lateinisch und griechisch), Hoch- und Tiefstellung sowie Runden möglich sein muss.

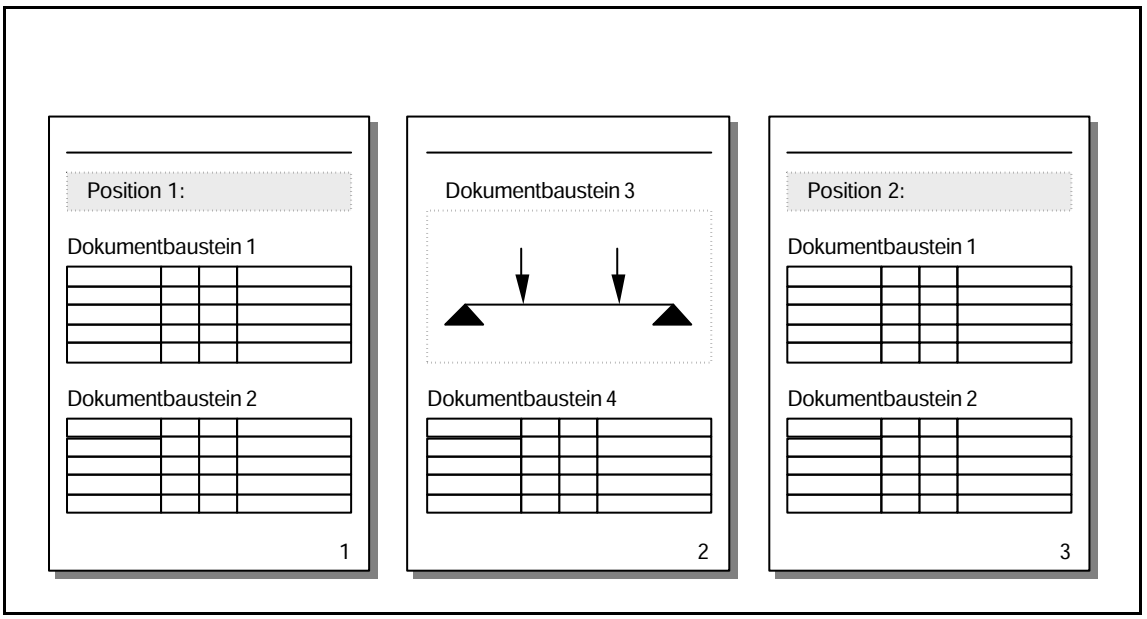


Abbildung 8.5: Tragwerksbericht mit Dokumenten und Dokumentbausteinen

Damit die Dokumente des Tragwerksberichtes und die darin enthaltenen Dokumentbausteine konsistent dargestellt werden, ist es zweckmäßig, dass sich die Darstellung der Dokumentbausteine (z.B. Schriftart, Schriftgröße, Zeilenabstand) selbständig der umgebenden Formatierung des Containers anpasst.

Die Datenhaltung der Dokumentbausteine soll auf dem im Rahmen von COM definierten Konzept des Structured Storage basieren. Damit können innerhalb einer Datei virtuelle Ordner angelegt werden, in denen beliebige Daten als binäre Streams abgelegt werden können. Dieses Verfahren wird genutzt, um die persistenten Eigenschaften der Dokumentbausteine direkt im als Container dienenden Word-Dokument zu speichern. Infolge der ermöglichten internen Ordnerstruktur kann jedes enthaltene Steuerelement innerhalb des sogenannten Object Pools über einen eigenen Unterordner verfügen. Darin können alle Daten ohne Interferenz zwischen den unabhängig voneinander entwickelten Dokumentbausteinen gespeichert werden (Abbildung 8.6).

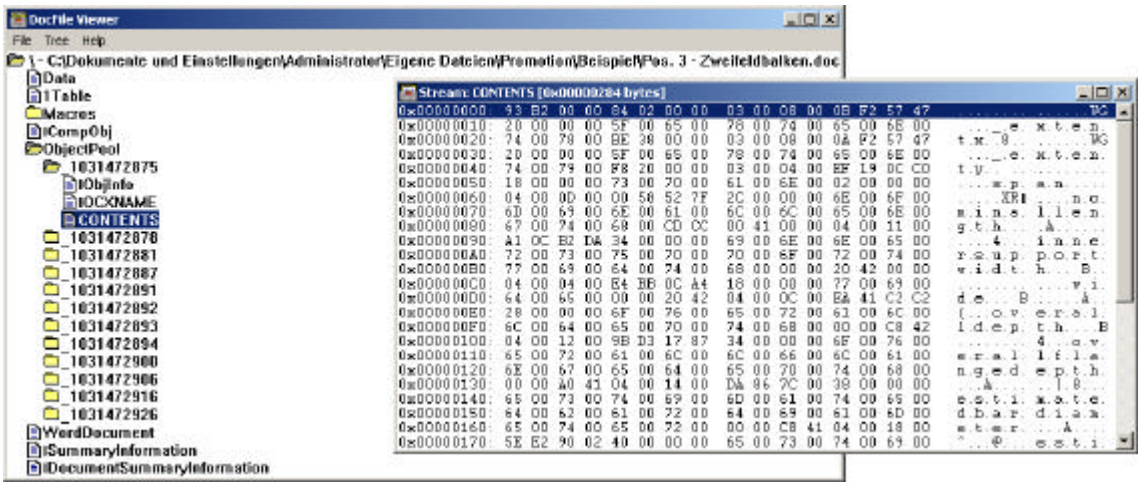


Abbildung 8.6: Datenhaltung in Verbunddateien (Konzept des Structured Storage)

Ursprünglich war vorgesehen, dass jeder Typ eines Dokumentbausteines durch eine eigenständige Software-Komponente ohne weitere Abhängigkeiten von anderen Software-Komponenten realisiert wird. Damit wäre den in Kapitel 5.2 formulierten Anforderungen an Software-Komponenten bezüglich Robustheit und Effizienz am besten entsprochen worden. Eine Machbarkeitsstudie hat jedoch ergeben, dass dieser Weg zu einem inakzeptabel hohen Codeanteil an nicht fachspezifischer Funktionalität führen würde. Die angestrebte Trennung von fachspezifischer und nicht-fachspezifischer Funktionalität wäre nicht erreicht worden, so dass die Entwicklung der Dokumentbausteine weiterhin nur durch Software-Entwickler möglich gewesen wäre. Da der nicht-fachspezifische Code im Wesentlichen auf die Tabellendarstellung entfällt und in ähnlicher Form in allen tabellenbasierten Dokumentbausteinen erforderlich ist, wird stattdessen der in Abbildung 8.7 dargestellten Architektur der Vorzug gegeben. Jedem tabellenbasierten fachspezifischen Dokumentbaustein wird dabei über den Mechanismus des Containments ein Tabellensteuerelement als innere Software-Komponente zugeordnet.

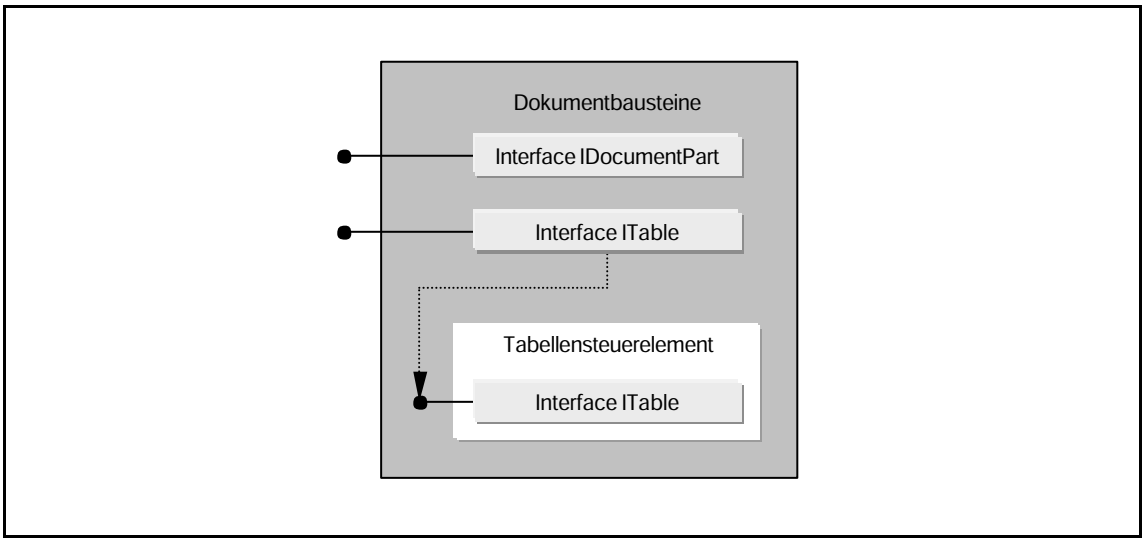


Abbildung 8.7: Dokumentbausteine mit über Containment integriertem Tabellensteuerelement

8.5 Entwurf der Bauteiltypen

Die diversen Dokumentbausteine für den Tragwerksbericht bilden feingranulare Software-Komponenten, die für jeweils einen konkreten Bauteiltyp zu einem grobgranularen Baustein zusammengefasst werden müssen. Zu diesem Zweck bietet die Microsoft Office-Plattform vier verschiedene Technologien an, die im Folgenden auf ihre Eignung hin untersucht werden:

- COM-Add-ins
- applikationsspezifische Add-ins
- Dokumentvorlagen
- Assistenten

8.5.1 Variante basierend auf COM Add-ins

Ein sogenanntes COM Add-in stellt eine spezielle Kategorie von Software-Komponenten dar, mit denen applikationsunabhängige Erweiterungen für alle Office-Anwendungen realisiert werden können. Die Entwicklung von COM Add-ins ist sprachneutral und kann somit in jeder Programmiersprache erfolgen, welche die Entwicklung von COM-Komponenten unterstützt. Die Implementierung wird kompiliert und in Form einer DLL verteilt. Daraus resultiert ein gutes Laufzeitverhalten, außerdem muss das Know-how zur Entwicklung des COM Add-ins nicht preisgegeben werden.

Nachteilig ist jedoch, das COM Add-ins wie alle anderen Software-Komponenten in der Windows-Registrierung eingetragen werden müssen. Das bedeutet, dass zur Installation eines Bauteiltyps ein spezielles Setup-Programm benötigt wird oder dass der Anwender die Registrierung manuell vornehmen muss. Hinzu kommt, dass ein COM Add-in nicht an ein konkretes Dokument gebunden ist. Folglich müssten zur Laufzeit die zur Bearbeitung eines Bauteiltyps benötigten Dokumentbausteine programmatisch in ein Dokument eingefügt werden. Das ist zwar ein gangbarer Weg, aber die visuelle Platzierung von Dokumentbausteinen in ein Dokument stellt eine wesentlich elegantere Lösung dar.

Insbesondere aufgrund des beträchtlichen Verwaltungsaufwandes wird die Implementierung von Bauteiltypen auf der Basis von COM Add-ins nicht als optimale Lösung angesehen.

8.5.2 Variante basierend auf applikationsspezifischen Add-ins

Die Funktionalität zur Bearbeitung der Bauteiltypen ist als Erweiterung speziell für Word als Containeranwendung vorgesehen. Folglich wird der Mehraufwand zur Realisierung eines applikationsunabhängigen Verhaltens nicht zwingend benötigt. Die Entwicklung von applikationsspezifischen Add-ins ist mit Visual Basic for Applications möglich. Die Implementierung wird in einer herkömmlichen Dokumentvorlage gespeichert. Diese kann durch den Anwender einfach in den Ordner kopiert werden, der für die Dokumentvorlagen des Tragwerkseditors vorgesehen ist. Da eine explizite Registrierung wie bei COM Add-ins nicht erforderlich ist, stellt dieser Ansatz auch für den Anwender eine deutliche Vereinfachung dar.

Nachteilig ist, dass die Entwicklung eines applikationsspezifischen Add-ins mit Visual Basic für Applications erfolgen muss. Der Quell-Code kann nicht kompiliert werden, sondern muss in einer Dokumentvorlage gespeichert werden. Darüber hinaus stehen applikationsspezifische Add-ins zwangsläufig allen momentan geöffneten Word-Dokumenten zur Verfügung. Dies ist nicht zweckmäßig, da die Nutzeroberfläche nur genau die Elemente enthalten sollte, die zur Bearbeitung des aktuellen Bauteiltyps benötigt werden. Sofern – wie es bei der Bearbeitung von Projekten üblich ist – mehrere Dokumente geöffnet sind, müssten auch gleichzeitig mehrere applikationsspezifische Add-ins aktiv sein.

Insbesondere aufgrund der daraus resultierenden möglicherweise verwirrenden Gestaltung der Nutzeroberfläche werden applikationsspezifische Add-ins als nicht optimal für die Implementierung der Bauteiltypen betrachtet.

8.5.3 Variante basierend auf Dokumentvorlagen

Als eine weitere Alternative bietet sich die Implementierung von Bauteiltypen auf der Basis von Dokumentvorlagen an. Damit können alle zur Bearbeitung von jeweils einem Bauteiltyp benötigten Dokumentbausteine visuell in einer Dokumentvorlage platziert werden. Die Dokumentvorlage kann außerdem Formatvorlagen, vorgefertigten Text, Erläuterungen und Skizzen enthalten und somit ein einheitliches Layout aller mit dem Tragwerkseditor erstellten Dokumente bewirken. Darüber hinaus kann der Anwender das Verbunddokument beliebig ergänzen und ändern. Der Informationsfluss zwischen den Dokumentbausteinen muss durch Scripte geregelt werden. Auch eine Anpassung von Menüs und Symbolleisten der Containeranwendung je nach bearbeitetem Bauteiltyp ist möglich.

Wie bei den applikationsspezifischen Add-ins ist das Kopieren von Dokumentvorlagen in den vorgesehenen Ordner ausreichend, so dass keine spezielle Registrierung erforderlich ist. Zur Laufzeit werden dem Anwender alle in diesem Ordner enthaltenen Dokumentvorlagen in einem Dialog zur Auswahl angeboten. Diese Vorgehensweise stellt für den Anwender die denkbar einfachste Variante dar.

Nachteilig ist wiederum, dass die Implementierung mit Visual Basic for Applications erfolgen muss und dass der Quell-Code in der Dokumentvorlage gespeichert wird. Da keine Kompilierung stattfindet, kann das Know-how zur Erstellung der Dokumentvorlage nicht zuverlässig geschützt werden. Dies stellt jedoch keine gravierende Einschränkung dar, da Scripte faktisch ohnehin nicht wiederverwendet werden können und somit auch nicht besonders schützenswert sind.

Infolge des geringen Aufwandes zur Realisierung und der einfachen Handhabung werden Bauteiltypen auf der Grundlage von Dokumentvorlagen als besonders zweckmäßige Lösung betrachtet.

8.5.4 Variante basierend auf Assistenten

Die zuvor diskutierten Dokumentvorlagen stellen ein sehr effizient zu nutzendes Hilfsmittel zur Erstellung komplexer technischer Dokumente dar. Es liegt jedoch ganz in der Verantwortung des Anwenders, alle relevanten Stellen des Dokuments zu identifizieren und die benötigten Informationen zu spezifizieren. Der Anwender muss sich vorab genau über die im Einzelnen notwendigen Schritte informiert haben.

Als Alternative bieten sich sogenannte Assistenten an, die dialoggesteuert in mehreren Schritten vom Anwender die benötigten Informationen erfragen und ihn zur Lösung führen. Infolge der vom Assistenten angebotenen zusätzliche Erläuterungen beschränkt sich der für den Anwender erforderliche Einarbeitungsaufwand auf ein Minimum (Abbildung 8.8). Assistenten können wahlweise auf der Basis von Add-ins oder von Dokumentvorlagen realisiert werden.

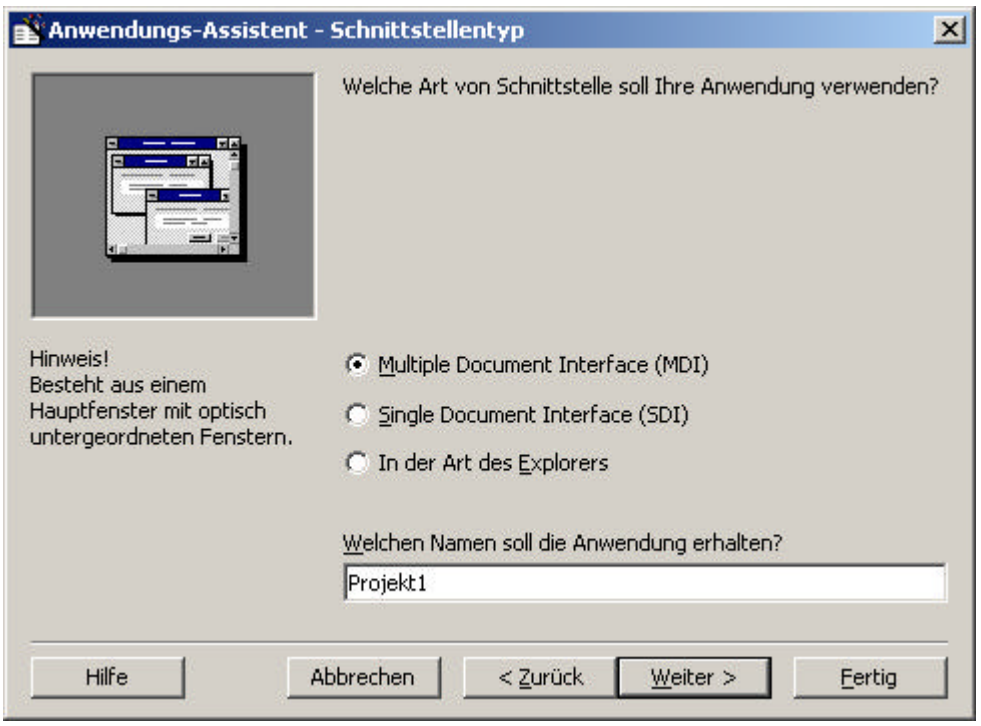


Abbildung 8.8: Beispiel eines Assistenten (Visual Basic Anwendungs-Assistent)

Problematisch ist jedoch, dass die Tragwerksplanung kein linearer Prozess ist, der anhand von vorab spezifizierten Parametern ohne den Eingriff des Tragwerksplaners ablaufen könnte. Stattdessen müssen die problembeschreibenden Parameter vom Tragwerksplaner Schritt für Schritt spezifiziert und die daraus resultierenden Ergebnisse sofort verifiziert werden können. In vielen Fällen kann unter Berücksichtigung der bereits spezifizierten Informationen keine zulässige oder zweckmäßige Lösung mehr ermittelt werden. Der Anwender muss dann gegebenenfalls mehrere Schritte zurückgehen, die relevanten Parameter identifizieren und variieren und den Prozess wiederholen.

Da es mit Assistenten in der bisher üblichen Form nicht möglich ist, Zwischenergebnisse anzuzeigen, scheinen sie zur Bearbeitung iterativer Prozesse nicht ohne weiteres geeignet zu sein. Aus diesen Gründen wird das Konzept an dieser Stelle nicht weiter verfolgt.

8.6 Entwurf der sonstigen Software-Komponenten

8.6.1 Gleichungslöser

Zur Realisierung der Anwendung des Tragwerkseditors wird an verschiedenen Stellen Funktionalität zur Lösung von mathematischen Problemen (beispielsweise Gleichungslöser) benötigt. Da der Performance bei einer nicht kommerziellen Anwendung keine besondere Bedeutung beigemessen wird, wird eine Implementierung des Eliminationsverfahren nach Gauß als ausreichend erachtet. Die Realisierung soll in Form einer Software-Komponente ohne Nutzeroberfläche erfolgen.

8.6.2 Schnittgrößenermittlung von Durchlaufträgern

Software zur numerischen Analyse von Stabtragwerken steht bereits in Form von Software-Komponenten zur Verfügung, so dass die Entwicklung einer eigenen Lösung nicht zwingend erforderlich ist. Die verfügbare Tragwerksplanungs-Software erlaubt jedoch nur die Analyse von Stabtragwerken auf der Grundlage der linearen Elastizitätstheorie. Das Tragverhalten von Massivbauwerken kann damit nur sehr ungenau beschrieben werden, da die Tragreserven von statisch unbestimmten Tragwerken unberücksichtigt bleiben. In der Realität kommt es jedoch – sofern sich ein Gleichgewichtszustand einstellen kann – vor dem Versagen des Tragwerks zur Bildung von Fließgelenken. Dadurch wird eine Umlagerung der Schnittgrößen bewirkt. Werden diese Tragreserven durch die Berücksichtigung des nichtlinearen Materialverhaltens in Ansatz gebracht, kann ein wesentlich gleichmäßigerer Verlauf der Bewehrung erreicht werden. Unter Inkaufnahme eines unerheblichen Mehraufwandes an Material kann dann auf eine sehr arbeitsaufwändige Staffelung der Bewehrung verzichtet werden. Da die Kosten von Massivbauwerken im Wesentlichen durch den Arbeitslohn bestimmt werden, stellt die somit ermöglichte rationelle Bewehrungsführung einen großen Vorteil dar.

Da die nichtlinearen Verfahren zur Schnittgrößenermittlung ein umfassendes theoretisches Verständnis erfordern und nur sehr aufwändig zu implementieren sind, soll stattdessen eine linear-elastische Berechnung mit begrenzter Momentenumlagerung durchgeführt werden. Bei Nachweis eines ausreichend duktilen Verhaltens (Nachweis der Rotationsfähigkeit), dürfen die Biegemomente um bis zu 30% reduziert werden. Aus diesen willkürlich festgelegten Schnittgrößen können unter Beachtung der Gleichgewichtsbedingungen die übrigen Schnittgrößen ermittelt werden. Diese werden maßgebend, sofern sie über den mit der linear-elastischen Berechnung ermittelten Werten liegen.

8.6.3 Integrationskomponente

Wie bereits im Kapitel 6.3.4 aus der Sicht der Tragwerksplaner beschrieben wurde, stellt ein Integrationskonzept ohne automatische Interpretation der Semantik der Informationen eine besonders ingenieurgemäße Lösung dar. Hinzu kommt, dass der Tragwerkseditor als unabhängig erweiterbares System – wie in Kapitel 7.2 erläutert – zwangsläufig auf einem Integrationsansatz beruhen muss, der wie die Integrationskonzepte ohne Interpretation der Semantik der Informationen unabhängig erweiterbar ist.

Um den Lastabtrag nach den Regeln des Eurocodes EC 1 so einfach und so nachvollziehbar wie möglich zu gestalten, werden die ebenfalls in Kapitel 6.3.4 diskutierten Vereinfachungen vorgenommen.

Der Tragwerkseditor basiert auf einer Tragwerksmodellierung mit dimensionsreduzierten Teiltragwerken und dem folgt dem Konzept der Dokumentzentrierung. Jedes Bauteil wird in einem Verbunddokument analysiert, nachgewiesen beziehungsweise bemessen und dokumentiert. Sämtliche Informationen werden innerhalb eines Dokumentes gespeichert, wobei der Informationsfluss innerhalb des Dokumentes durch Scripte geregelt wird. Eine Integration innerhalb der Dokumente ist somit gegeben.

Bestimmte Informationen müssen jedoch zwischen den Dokumenten eines Projektes ausgetauscht werden. In vielen Fällen stellen die Ergebnisse der Analyse einer Position die Ausgangswerte zur Berechnung einer anderen Position dar. Beispielsweise entsprechen die Auflagerreaktionen eines Bauteils den Einwirkungen des darunter liegenden Bauteils. Für die effiziente und fehlerunanfällige Übernahme dieser Informationen wird eine Integrationskomponente benötigt. Diese Komponente soll die Integration von skalaren Größen und Zeichenketten zwischen den zu einem Projekt gehörenden Dokumenten ermöglichen und dabei auf eine Interpretation der Semantik der Informationen verzichten.

Zur besseren Unterscheidung werden nachfolgend Dokumente, aus denen Informationen entnommen werden, als Quelldokumente bezeichnet. Entsprechend werden Dokumente, in die Informationen eingefügt werden, als Zieldokumente bezeichnet.

Der Ablauf der Integration erfolgt nach dem folgenden Schema: Nachdem der Anwender eine in einem Quelldokument erzeugte Information selektiert hat, kann er wahlweise durch die Tastenkombination Strg + C, über das Menü Bearbeiten und den Menüpunkt Kopieren oder über das Symbol Kopieren den Wert der Information in die Zwischenablage kopieren. Danach muss er im entsprechenden Zieldokument die gewünschte Einfügestelle spezifizieren und über das Menü Bearbeiten und den Menüpunkt Inhalte einfügen ... den Wert der Information aus der Zwischenablage einfügen. Im Hintergrund protokolliert die Integrationskomponente die folgenden Informationen:

- Ordner
- Dokumentnamen
- Identifikatoren der Dokumentbausteine
- Zeilennummern
- Spaltennummern

Die Informationen müssen sowohl für das Quell- als auch das Zieldokument gespeichert werden. Anhand dieser Angaben können die Informationen in Quell- und Zieldokumenten eindeutig identifiziert werden.

Zur Datenhaltung bietet sich wegen der tabellenorientierten Informationsstruktur eine relationale Datenbank an. Diese dient zur Speicherung der Entnahme- und Einfügestellen der Informationen. Mit Hilfe dieser Angaben können Änderungen erkannt werden, indem die Information im Quelldokument mit der Information im Zieldokument verglichen wird.

Der Datenbestand kann unter zwei wesentlichen Gesichtspunkten ausgewertet werden. Mit Hilfe der in der Datenbank gespeicherten Informationen kann erkannt werden, welche Zieldokumente von der Änderung eines Quelldokuments betroffen sind. Umgekehrt kann auch überprüft werden, aus welchen Quelldokumenten Informationen in ein Zieldokument übernommen wurden. Um eine ingenieurgemäße Arbeitsweise zu ermöglichen, müssen auch inkonsistente Zustände möglich sein. Damit der Anwender nicht bei jeder Überprüfung der Konsistenz erneut gefragt wird, ob er einen bewusst inkonsistent gelassenen Zustand beibehalten möchte, sollte diese Einstellung als zusätzliches Attribut in der Datenbank vermerkt werden.

Alternativ zu dieser Vorgehensweise wäre auch eine zusätzliche Speicherung der Werte der Attribute möglich. Das hätte den Vorteil, dass Änderungen allein durch Auswertung der Datenbank erkannt werden könnten, ohne dass dazu alle Dokumente vorliegen und geöffnet sein müssen. Nachteilig ist jedoch, dass jede Änderung der integrierten Informationen eine Aktualisierung der Datenbank erfordern würde. Da die Performance während der Bearbeitung der Dokumente beeinträchtigt würde, wird auf eine Speicherung der Werte der Informationen verzichtet.

Innerhalb eines Projektes können alle zur Integration benötigten Informationen in einer Tabelle der Datenbank mit folgendem Datenbankschema gespeichert werden.

Attribut	Datentyp	Kommentar
Source Folder	String	Ordner des Quelldokuments
Source Document	String	Name des Quelldokuments
Source Control	String	Name des Quellsteuerelements
Source Row	Long	Zeilennummer im Quellsteuerelement
Source Column	Long	Spaltennummer im Quellsteuerelement
Target Folder	String	Ordner des Zieldokuments
Target Document	String	Name des Zieldokuments
Target Control	String	Name des Zielsteuerelements
Target Row	Long	Zeilennummer im Zielsteuerelement
Target Column	Long	Spaltennummer im Zielsteuerelement
Consistency	Boolean	Konsistenzschalter

Tabelle 8.1: Datenbankschema der Integrationskomponente

Das entwickelte Integrationskonzept ohne Interpretation der Semantik der Informationen stellt eine semi-automatische Lösung dar. Die benötigten Verknüpfungen zwischen den Positionen müssen einmal durch den Anwender spezifiziert werden. Damit wird eine größtmögliche Flexibilität erreicht und dem Tragwerksplaner der volle Gestaltungsspielraum erhalten. Die eigentliche Übernahme der Information erfolgt dann ohne weiteres Zutun des Anwenders. Bei Änderungen können Inkonsistenzen automatisch erkannt und auf Wunsch des Anwenders auch automatisch aufgehoben werden.

8.7 Gesamtübersicht der Architektur des Tragwerdeditors

Die entworfene Architektur gliedert sich in die Anwendung des Tragwerkseditors, die Dokumentbausteine, die Bauteiltypen und einige sonstige Software-Komponenten. Eine Gesamtübersicht ist in Abbildung 8.9 dargestellt.

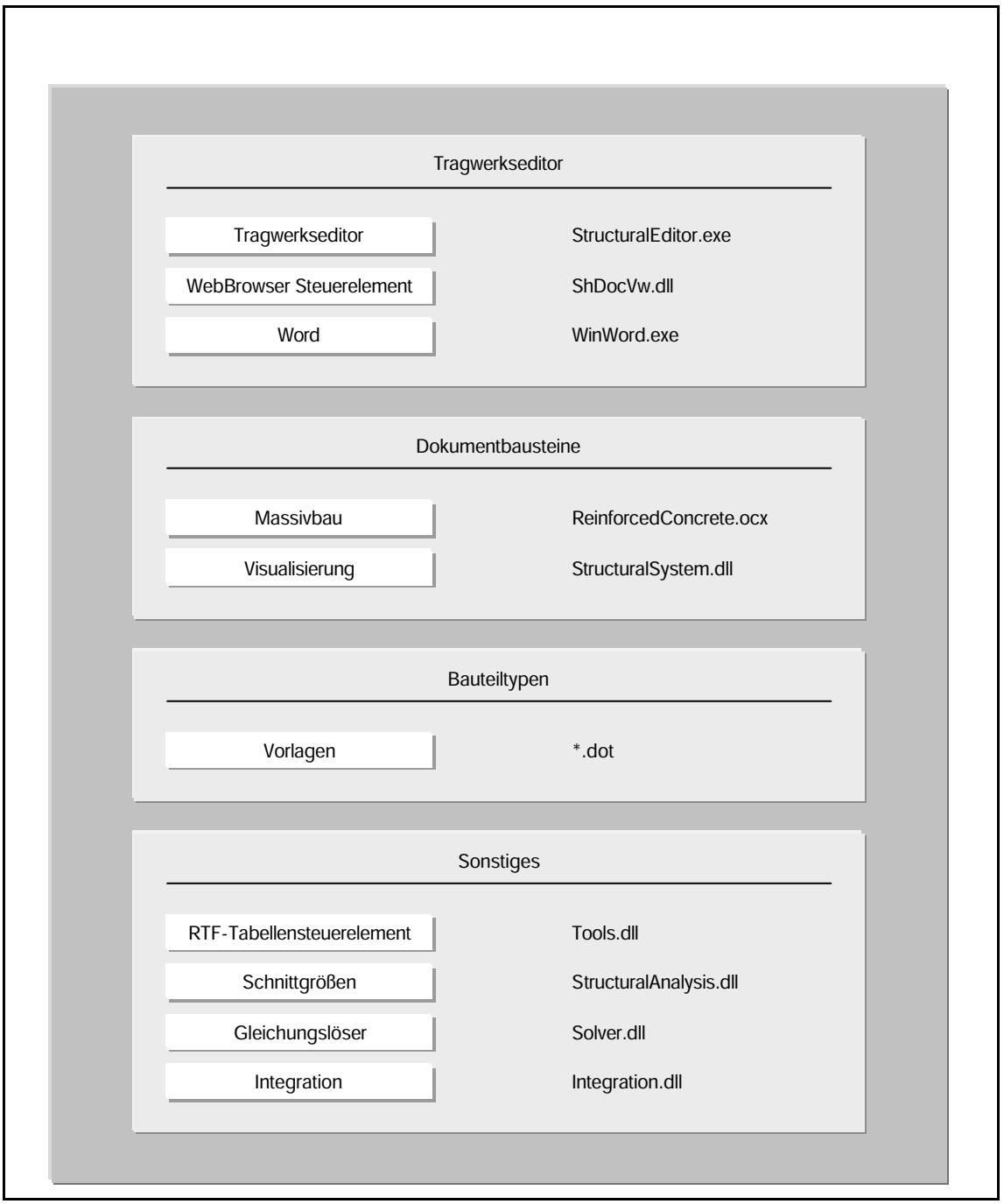


Abbildung 8.9: Gesamtübersicht der Komponenten des Tragwerkseditors

Kapitel 9
Realisierung der Pilotimplementierung

In diesem Kapitel wird der Entwurf der Architektur des Tragwerkseditors aus dem vorangegangenen Kapitel mit einer Pilotimplementierung verifiziert. Dabei wird detailliert auf die eingesetzten Technologien und Komponenten eingegangen. Für jede Komponente werden Realisierungsaufwand sowie gegebenenfalls aufgetretene Schwierigkeiten und Schlussfolgerungen für eine verbesserte Implementierung angegeben.

9.1 Allgemeines

Die Umsetzung der in den vorangegangenen Kapiteln erläuterten Konzepte erfolgt insbesondere wegen des umfassenden Angebotes an Standard-Software auf der Windows-Plattform. Als Komponentenmodell wird das Component Object Model (COM) eingesetzt. Eine mögliche Plattformunabhängigkeit als besondere Stärke von Java und CORBA wird nicht als vordergründiges Entwicklungsziel angesehen.

Als Programmiersprache für die Realisierung der Pilotimplementierung wird Visual Basic eingesetzt, da gegenwärtig damit die Vorteile der komponentenorientierten Software-Entwicklung, wie Verringerung des Entwicklungsaufwandes und Vereinfachung der Software-Entwicklung, am besten ausgenutzt werden können. Aus dem Einsatz von Visual Basic als Rapid-Prototyping-System resultieren einige Einschränkungen bezüglich des Leistungsumfangs und in Teilbereichen der Performance. Diese Einschränkungen können bei einer Pilotimplementierung toleriert werden.

Im Unterschied dazu hätte sich die komponentenbasierte Entwicklung in C++ zur Zeit extrem aufwändig und komplex gestaltet, wodurch die Vorteile der Wiederverwendung teilweise aufgehoben worden wären. Mit Java könnte ohne Verwendung der Microsoft-Variante Visual J++ und den damit verbundenen Microsoft-spezifischen Erweiterungen nicht auf die Funktionalität von Standard-Software zugegriffen werden.

An Standard-Software wird Windows 2000 Service Pack 1 (einschließlich Internet Explorer 5.5) und Word 2000 Service Release 1 eingesetzt. Die Entwicklung erfolgt mit Visual Basic 6 Service Pack 4

9.2 Implementierung des Tragwerkseditors

Wie bereits im Kapitel 8.3 festgestellt wurde, ist die Technologie der Active Documents zur Einbindung von kompletten Dokumenten wesentlich besser geeignet als die Technologie der OLE Objects. Dementsprechend soll der selbst entwickelte Tragwerkseditor aus einer Rahmenanwendung bestehen, die als Active Document Container Word-Dokumente einbindet. Word fungiert dabei als Active Document Server.

Jeder Active Document Container muss über Implementierungen der Interfaces IOleDocumentSite, IContinueCallback und IOleCommandTarget verfügen. Diese werden durch Visual Basic nicht bereitgestellt. Von der prinzipiell möglichen Eigenimplementierung dieser Interfaces wird wegen des unverhältnismäßig hohen Entwicklungsaufwandes abgesehen.

Als Behelf kommt das zum Lieferumfang von Visual Basic gehörende OLE-Container Steuerelement in Betracht. Dieses erlaubt genau genommen nur die Einbettung eines OLE Objects. Dennoch ist mit diesem Steuerelement auch die Bearbeitung von kompletten Word-Dokumenten möglich, wobei Menüs automatisch gemischt und Symbolleisten ersetzt werden können. Die Bearbeitung größerer Dokumente ist über Bildlaufleisten möglich. Aufgrund der mit der expliziten Aktivierung verbundenen komplizierteren Nutzerführung und der völlig unzureichenden Stabilität und Performance dieser Software-Komponente wird von einem Einsatz abgesehen. Als Alternative bietet sich die Einbindung von Komponenten des Internet Explorers an. Dessen Struktur stellt ein Beispiel für eine ausgeprägte Komponentenarchitektur dar (Abbildung 9.1).

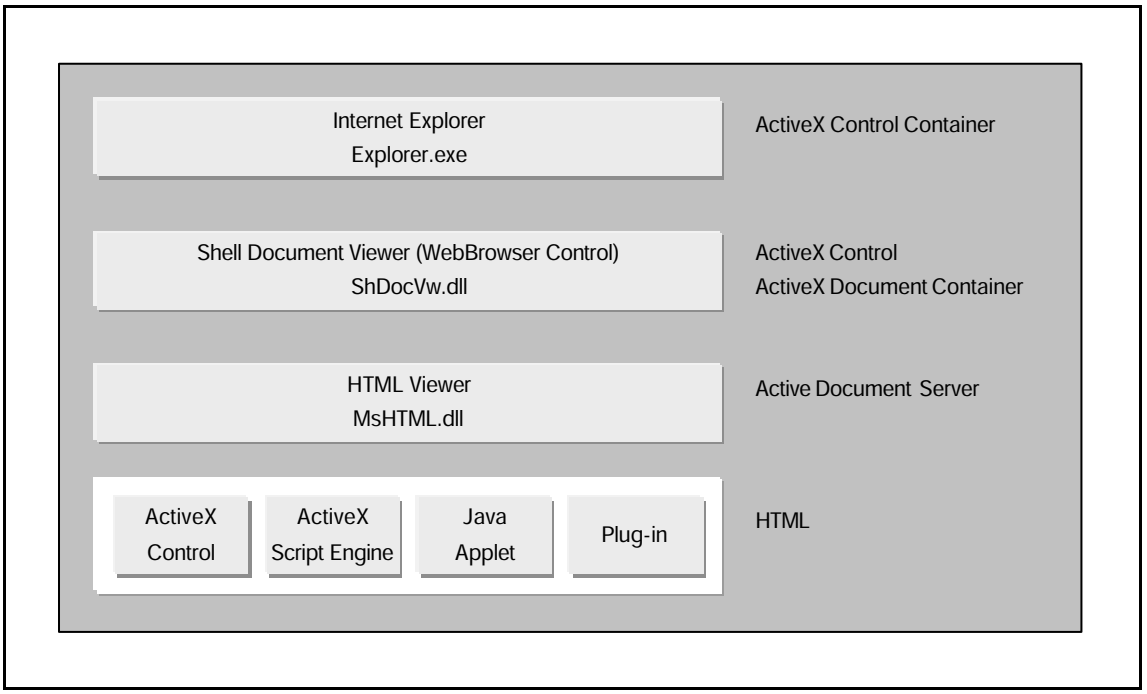


Abbildung 9.1: Architektur des Internet Explorers

Die eigentliche Anwendung ist ein ActiveX Control Container, der nur für die Anzeige des Rahmenfensters und die Ereignisverwaltung verantwortlich ist. Die gesamte Funktionalität zur Interpretation und Darstellung von HTML-Code ist als separate Software-Komponente ausgegliedert worden, welche die Merkmale eines Active Document Servers aufweist. Die Verbindung zwischen diesen beiden Bausteinen wird durch den sogenannten Shell Document Viewer hergestellt, der sowohl ein ActiveX Control (WebBrowser Control) als auch einen Active Document Container darstellt. Somit stellt die Darstellung von HTML-Seiten im Internet Explorer nur einen Sonderfall dar, ebenso gut können Active Documents fremder Anwendungen eingebunden werden.

Die beschriebene Architektur ermöglicht es externen Software-Entwicklern, mit jeder COM-fähigen Programmiersprache nahezu die gesamte Funktionalität des Internet Explorers wiederzuverwenden. Da jedes Visual Basic Form einen ActiveX Control Container darstellt, kann das WebBrowser-Control auf denkbar einfache Weise eingebunden werden. Damit können Active Document Container auch in Visual Basic realisiert werden. Im konkreten Fall wird dieser Sachverhalt genutzt, um Word-Dokumente in die Anwendung des Tragwerkseditors zu integrieren. Das WebBrowser Control zeichnet sich durch ein sehr zuverlässiges und ressourcensparendes Laufzeitverhalten aus. Aufgrund der zentralen Bedeutung des Internet Explorers kann mit einer intensiven Wartung und Weiterentwicklung gerechnet werden.

Der Einsatz unter Visual Basic ist jedoch mit Einschränkungen verbunden. Beispielsweise ist es nicht möglich, die Menüs des Active Document Servers anzuzeigen, sie müssen deshalb selbst implementiert werden. Darüber hinaus können Symbolleisten nicht im MDI-Parent-Window angezeigt werden. Da eine gegebenenfalls mehrfache Darstellung in den MDI-Child-Windows nicht konform mit dem Ziel einer möglichst ergonomischen Nutzeroberfläche ist, werden die in der Voreinstellung permanent angezeigten Symbolleisten "Standard" und "Format" selbst implementiert.

Diese Einschränkung verursacht einen erheblichen Mehraufwand. Zunächst müssen die Menüs und Symbolleisten im Tragwerkseditor selbst angelegt werden. Außerdem müssen Ereignisroutinen für alle Menüpunkte und Symbole implementiert werden. Dies ist mit verhältnismäßig geringem Aufwand möglich, da nahezu die gesamte Funktionalität von Word über COM-Interfaces verfügbar gemacht wird. Allerdings haben zahlreiche Aktionen des Anwenders im Word-Dokument Auswirkungen auf die Darstellung der Nutzeroberfläche. Beispielsweise muss nach jedem Bewegen der Einfügemarke der Status aller Menüpunkte und Symbole überprüft werden. Infolgedessen kann der Tragwerkseditor nicht auf einer herkömmlichen Client-Server-Architecture basieren, bei der die Methoden des Servers (Word) in der vom Client (Tragwerkseditor) festgelegten Reihenfolge aufgerufen werden. Stattdessen müssen der Tragwerkseditor als Active Document Container und Word als Active Document Server interaktiv zusammenarbeiten. Das heißt, sämtliche relevanten Ereignisse des Servers müssen unter Nutzung eines Notifikationskonzeptes an die Rahmenanwendung weitergeleitet werden und dort entsprechende Reaktionen veranlassen. Um eine veränderte Position der Einfügemarke zu signalisieren, wird von Word das Interface IApplicationEvents2 bereitgestellt, dessen Bestandteil unter anderem die Methode WindowSelectionChange ist (Abbildung 9.2).

```
Interface IApplicationEvents2 : IDispatch
{
     ...
     HRESULT WindowSelectionChange(Selection* Sel);
     ...
};
```

Abbildung 9.2: Interface IApplicationEvents2 mit der Methode WindowSelectionChange

Der verfolgte Ansatz ermöglichte die Erstellung eines Prototypen des Tragwerkseditors mit einem äußerst geringen Entwicklungsaufwand von circa 2.800 Codezeilen. Durch die Entwicklung einer eigenständigen Anwendung bestanden alle Freiheiten, auf die speziellen fachspezifischen Erfordernisse zu reagieren. Als wesentlicher Unterschied zu Standard-Textverarbeitungen wurde exemplarisch eine Projektverwaltung realisiert. Die Projektdaten werden in einer separaten Projektdatei gespeichert und mit einem Strukturansicht-Steuerelement dargestellt (Abbildung 9.3). Infolge des auf Software-Komponenten basierenden Entwicklungsansatzes können Rahmenanwendung und Word unabhängig voneinander weiterentwickelt werden. Beispielsweise war der Übergang von Word 97 auf Word 2000 ohne Änderung der Implementierung oder Neukompilierung möglich.

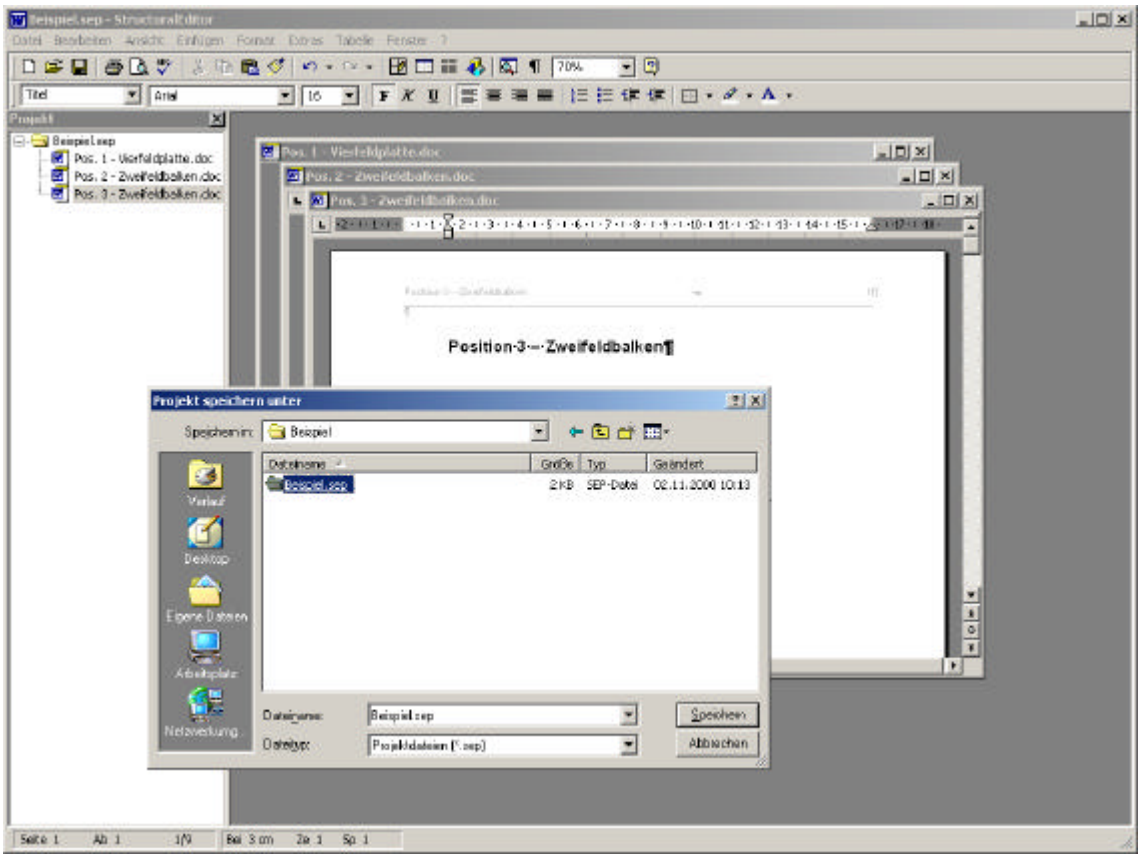


Abbildung 9.3: Prototyp des Tragwerkseditors

9.3 Implementierung der Dokumentbausteine

9.3.1 RTF-Tabellensteuerelement

Die effizienteste Möglichkeit zur Realisierung des tabellenbasierten Steuerelements wäre der Einsatz eines der sogenannten FlexGrid Steuerelemente gewesen. Alle bekannten Steuerelemente dieser Art unterstützen jedoch nur die Anzeige von unformatiertem Text, so dass die für Tragwerksberichte typischen Darstellungen nicht angezeigt werden können.

Das einzige Steuerelement, welches über Funktionalität für formatierte Texte (Rich Text Format) verfügt und zum Lieferumfang des Betriebssystems oder der Entwicklungsumgebung gehört, ist das Rich Textbox-Steuerelement. Dabei handelt es sich jedoch um ein einzelnes Steuerelement, das nicht zu einer Tabelle zusammengefügt werden kann. Infolgedessen ist die Entwicklung eines eigenen Steuerelementes notwendig. Der dafür erforderliche Aufwand ist sehr hoch und entspricht nahezu der Entwicklung eines RTF-Texteditors. Die Implementierung des Rich Textbox-Steuerelements stützt sich im Wesentlichen auf die Rich Edit Bibliothek. Leider wird diese Funktionalität nicht in Form einer Software-Komponente bereitgestellt, sondern als dynamisch bindbare Bibliothek (DLL). Prinzipiell könnte ein RTF-Tabellensteuerelement unter Wiederverwendung dieser DLL entwickelt werden. Der dafür erforderliche Entwicklungsaufwand ist jedoch für einen Prototypen immer noch inakzeptabel hoch.

Im Rahmen der Pilotimplementierung wird deshalb ein RTF-Tabellensteuerelement entwickelt, bei dem für jede Tabellenzelle über den Mechanismus des Containments ein Rich Textbox-Steuerelement benutzt wird (Abbildung 9.4). Der Nachteil dieses Ansatzes besteht darin, dass bei den zur Dokumentation einer Position üblichen Dokumentgrößen durchaus mehrere Tausend Instanzen des Rich Textbox-Steuerelements benötigt werden. Damit ist ein erheblicher Ressourcenverbrauch verbunden. Die dadurch bedingten Performance-Einbußen müssen in einer kommerziellen Version optimiert werden.

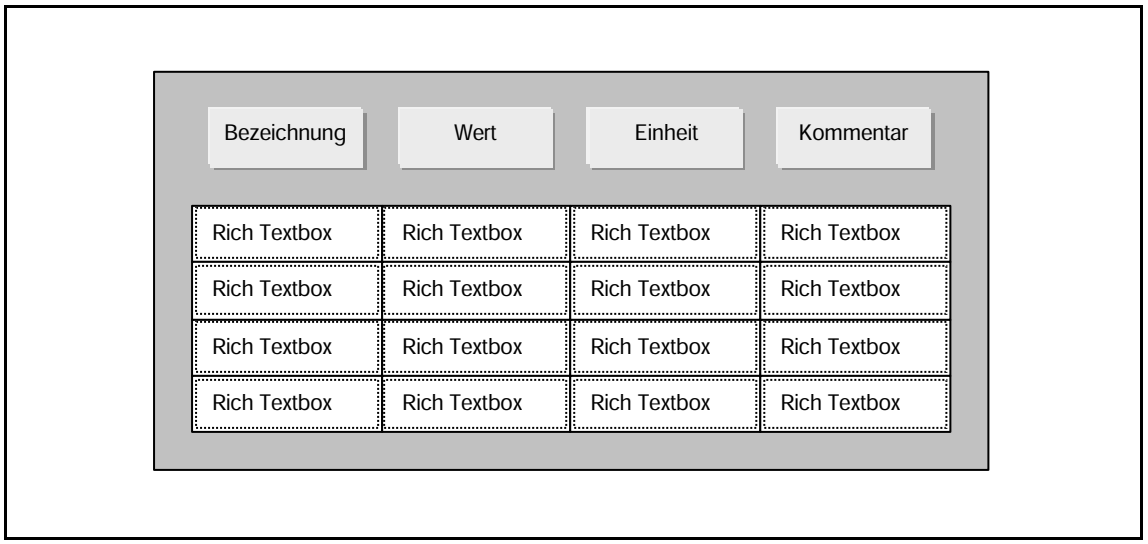


Abbildung 9.4: Aufbau des RTF-Tabellensteuerelements

Die Größe des RTF-Tabellensteuerelements wird aus den spezifizierten Spaltenbreiten sowie der Zeilenanzahl und Schriftgröße bestimmt. Hinzu kommt, dass eine Textverarbeitung mit einem flexiblen Zoom arbeitet. Das heißt, ein Steuerelement hat nicht wie sonst üblich eine feste Größe. Stattdessen muss die Größe permanent dem aktuellen Zoom-Faktor der Textverarbeitung entsprechend angepasst werden. Eine Größenanpassung des RTF-Tabellensteuerelements ist außerdem bei jeder Änderung der Größe des MDI-Child-Windows erforderlich. Aufgrund der großen Anzahl der davon betroffenen Instanzen des Rich Textbox-Steuerelements ist dies ein kritischer Aspekt für die Performance.

Je nach Wunsch des Anwenders kann das RTF-Tabellensteuerelement in verschiedenen Rasterdarstellungen (Tabelle, Kommentarlinie, ohne Raster) angezeigt werden. In dieses Raster werden die Instanzen des Rich Textbox-Steuerelementes eingepasst.

Entsprechend der geforderten konsistenten Darstellung muss die Präsentation des RTF-Tabellensteuerelements bei jeder Änderung von Schriftart und -größe im Container angepasst werden. Diese Änderung kann nicht global für ein komplettes RTF-Tabellensteuerelement erfolgen. Bei einer Änderung der Schriftart dürfen nur die lateinischen Buchstaben in der neuen Schriftart dargestellt werden, wohingegen alle griechischen Buchstaben unverändert bleiben müssen. Gleichermaßen darf eine geänderte Schriftgröße nur den in normaler Schriftgröße gesetzten Zeichen zugewiesen werden, während für Hoch- und Tiefstellungen eine reduzierte Schriftgröße festgelegt werden muss. Infolgedessen wird ein RTF-Parser benötigt, der den RTF-Code der einzelnen Rich Textbox-Steuerelemente interpretiert und entsprechend den Änderungen in der Umgebung des Containers modifiziert.

Aus den genannten Gründen ist die Entwicklung des RTF-Tabellensteuerelements verhältnismäßig aufwändig. Die Ausgliederung in ein separates Steuerelement ist in jedem Fall zweckmäßig. Die daraus resultierenden engen Abhängigkeiten zwischen Dokumentbausteinen und RTF-Tabellensteuerelement müssen in Kauf genommen werden.

Der Umfang der Pilotimplementierung des RTF-Tabellensteuerelements liegt bei circa 700 Codezeilen. Diese Zahl sagt jedoch wenig über den tatsächlich betriebenen Entwicklungsaufwand aus. Bereits der Entwurfsprozess gestaltete sich sehr aufwändig, da zahlreiche wechselseitige Abhängigkeiten zwischen RTF-Tabellensteuerelement, Dokumentbausteinen und dem Tragwerkseditor zu beachten waren. Darüber hinaus sind viele Details der Spezifikation von ActiveX-Steuerelementen und -Containern schlecht dokumentiert oder wurden davon abweichend implementiert. Erschwerend kommt hinzu, dass ein Debugging der Steuerelemente nur innerhalb der Entwicklungsumgebung im Quell-Code möglich ist. Viele Aspekte von Steuerelementen können jedoch nur in den anvisierten Containeranwendungen getestet werden, wo lediglich eine Disassemblierung möglich ist. Der Großteil des Entwicklungsaufwandes lag folglich nicht in der eigentlichen Implementierung, sondern in der Evaluierung der eingesetzten Software-Komponenten, dem iterativen Entwurf der Architektur und dem Debugging. Da die Implementierung des Prototypen des RTF-Tabellensteuerelements auf die essentiellen Funktionsmerkmale beschränkt wurde, muss bei der Realisierung einer kommerziellen Version ein Vielfaches des bisherigen Entwicklungsaufwandes erwartet werden.

9.3.2 Tabellenbasierte Dokumentbausteine für den Massivbau

Entsprechend dem in Kapitel 10 erläuterten Beispiel werden 10 Dokumentbausteine für den Massivbau nach Eurocode EC 2 entwickelt. Die Aufteilung des Funktionsumfangs in verschiedene Dokumentbausteine orientiert sich an den genormten Nachweisen. Bei der Entwicklung wurde auf eine möglichst hohe Flexibilität der Dokumentbausteine Wert gelegt.

Zum Beispiel erlauben die Dokumentbausteine zur Biegebemessung sowohl die Verwendung von Stabstahl als auch von Mattenbewehrung. Bedingung ist jedoch eine konstante Zeilenzahl, da sich andernfalls die Komplexität der Implementierung sehr stark erhöhen würde. Folglich werden bei ähnlichen Aufgabenstellungen, die eine unterschiedliche Zeilenzahl erfordern (z.B. Berechnung der Schnittgrößen von Zweifeld- und Vierfeldträgern) zwei unabhängige Dokumentbausteine implementiert.

Die Steuerung der Dokumentbausteine erfolgt vollständig über Dialogboxen. Alternativ wäre auch eine direkte Eingabe der Parameter in Zellen der Dokumentbausteine möglich gewesen. Mit Hilfe der Dialogboxen können jedoch die Nutzereingaben geprüft und gegebenenfalls zurückgewiesen werden. Um die teilweise große Zahl von Parametern zu gruppieren und damit übersichtlich zu präsentieren, benutzen die Dialogboxen Registerkarten. Das Design der Registerkarten folgt einheitlichen Gestaltungsrichtlinien. Wenn verschiedene Dialoge über die gleichen Registerkarten verfügen, bleibt deren Reihenfolge stets gleich (Abbildung 9.5).

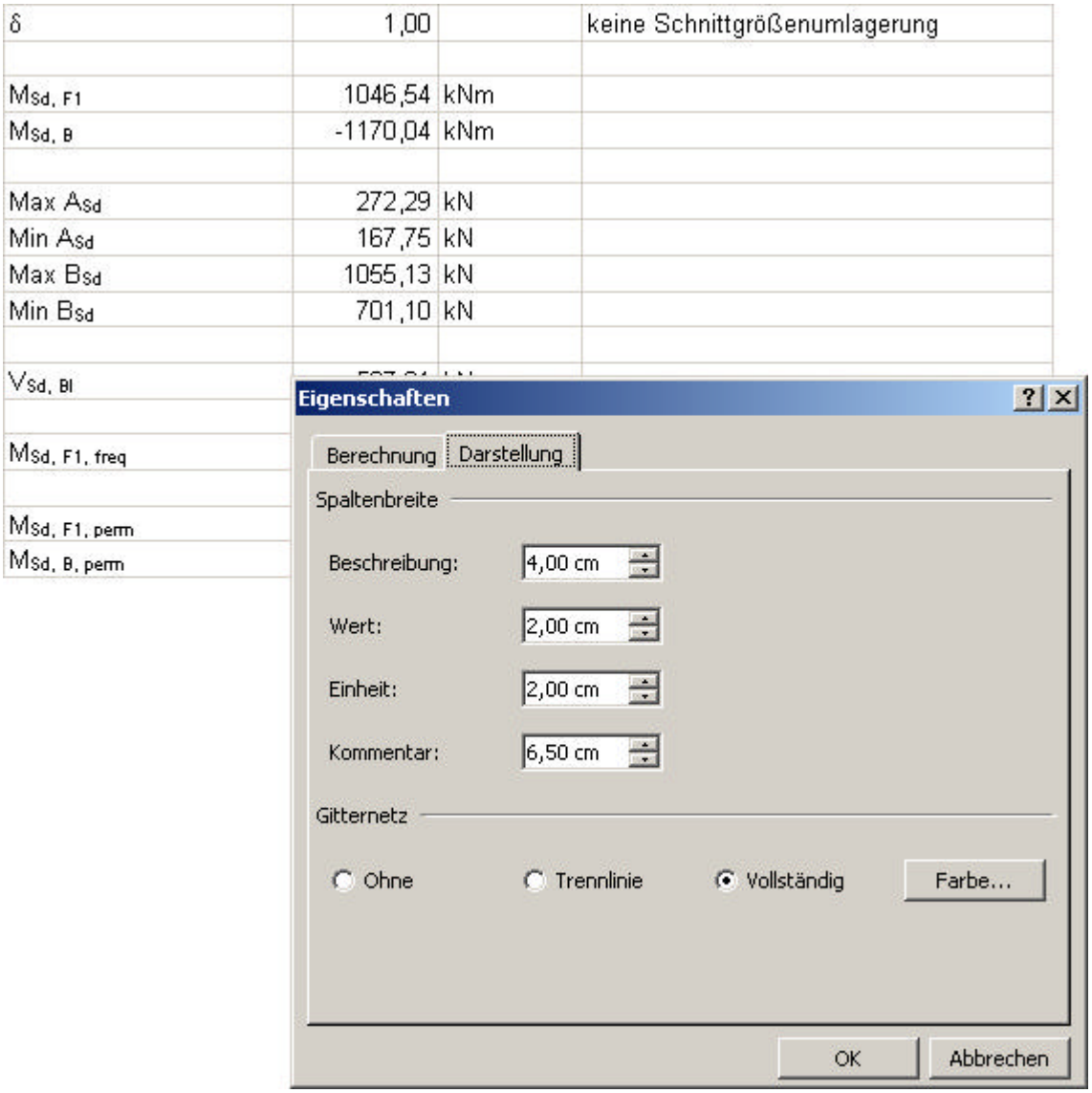


Abbildung 9.5: Dokumentbaustein zur Berechnung der Schnittgrößen von Zweifeldträgern

Die Größe der Dokumentbausteine kann vom Anwender nicht direkt festgelegt werden, sondern ist nur indirekt durch eine Änderung der Spaltenbreiten oder der Schriftgröße des enthaltenen RTF-Tabellensteuerelements möglich. Sofern die Dokumentbausteine sich in Word als Containeranwendung befinden, werden alle Resize-Ereignisse ausgewertet und der aktuelle Zoom-Faktor abgefragt. Der ermittelte Zoom-Faktor wird an das über Containment enthaltene RTF-Tabellensteuerelement weitergeleitet, welches daraus seine Größe berechnen kann. Durch Auslösung eines Ereignisses wird die vom RTF-Tabellensteuerelement ermittelte Größe an den Dokumentbaustein zurückgegeben. Der Dokumentbaustein passt sich daraufhin der Größe des RTF-Tabellensteuerelements an. Um unbeabsichtigte oder willkürlich vorgenommene Modifikationen der Dokumentbausteine zu verhindern, sind die ersten drei Spalten für Bezeichnung, Wert und Einheit gesperrt. Kommentare können hingegen jederzeit in der entsprechenden Spalte vom Anwender eingetragen werden. Die eingefügten Kommentare werden gemeinsam mit den übrigen Eigenschaften der Dokumentbausteine im Word-Dokument persistent gespeichert.

Dokumentbaustein	Aufgabe	Codezeilen
Geometry	Spezifikation der Geometrie	1.800
Material	Spezifikation der Materialien	1.100
Action	Spezifikation der Einwirkungen	1.800
InternalForces2Span	Linear-elastische Berechnung der Schnittgrößen von Zweifeldträgern mit optionaler Umlagerung	1.100
InternalForces4Span	Linear-elastische Berechnung der Schnittgrößen von Vierfeldträgern mit optionaler Umlagerung	1.300
BendingDesignIBeam	Biegebemessung von I-Querschnitten nach dem Berechnungsmodell mit Betonspannungsblock	1.200
BendingDesignTBeam	Biegebemessung von T-Querschnitten nach dem Berechnungsmodell mit Betonspannungsblock	1.300
ShearDesignPlate	Schubbemessung von Platten nach dem Berechnungsmodell mit konstanter Druckstrebenneigung	900
ShearDesignBeam	Schubbemessung von Balken nach dem Berechnungsmodell mit konstanter Druckstrebenneigung	1.700
Deflection	Nachweis der Verformungen über die Einhaltung von Bauteilschlankheiten	900
Gesamt		**13.100**

Tabelle 9.1: Dokumentbausteine für den Massivbau

Der Entwicklungsaufwand für die Dokumentbausteine ist mit insgesamt 13.100 Zeilen sehr hoch, insbesondere wenn der prototyphafte Charakter und der vergleichsweise niedrige Aufwand für die übrige Pilotanwendung berücksichtigt werden (Tabelle 9.1). Die Ursache liegt zum einen darin, dass neben dem Rechenablauf verhältnismäßig viel Verwaltungsaufwand benötigt wird. Neben der Spezifikation der Interfaces müssen die Zugriffsmethoden für die Eigenschaften implementiert werden und es muss festgelegt werden, welche Eigenschaften persistent sein sollen. Vorteilhaft ist jedoch, dass alle tabellenbasierten Dokumentbausteine nach denselben Prinzipien aufgebaut werden können. Darüber hinaus ist es durch die Aufteilung in fachspezifische Dokumentbausteine und ein generisches RTF-Tabellensteuerelementes gelungen, die software-technischen und die fachlichen Aspekte weitgehend voneinander zu trennen. Weiterhin kann die Erstellung von zusätzlichen Dokumentbausteinen nach demselben Schema erfolgen. Die Implementierung eines weiteren Dokumentbausteins gestaltet sich somit verhältnismäßig einfach und ist trotz des großen Umfangs der Implementierung in sehr kurzer Zeit möglich (vergleiche Kapitel 10.3.1).

Auffällig ist, dass insbesondere die Erstellung der Dialogboxen sehr aufwändig ist. Fast die Hälfte des gesamten Implementierungsaufwandes der Dokumentbausteine wurde durch die Dialogboxen verursacht. In einigen Fällen konnten Dialogboxen jedoch für ähnliche Dokumentbausteine wiederverwendet werden.

Beim Testen der Dokumentbausteine haben sich unerwartete Probleme mit der Druckqualität ergeben. Entsprechend der Spezifikation müssen alle Steuerelemente mit voller Druckerauflösung ausgeben werden. Die auf dem RTF-Tabellensteuerelement basierenden Dokumentbausteine werden jedoch im Ausdruck nur mit einer reduzierten Auflösung dargestellt.

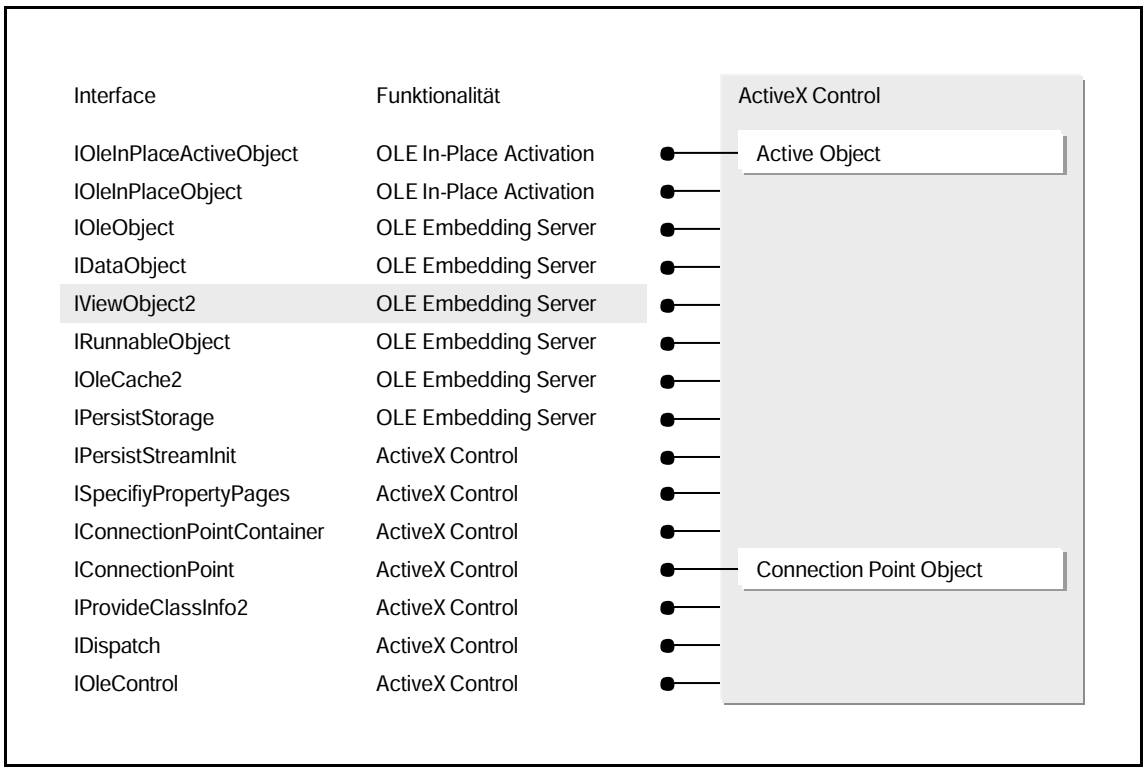


Abbildung 9.6: Interfaces eines ActiveX-Controls (Chappell, 1996)

Die Ursache dafür liegt vermutlich in dem vom Rich Textbox-Steuerelement fehlerhaft implementierten Interface IViewObject2 (Abbildung 9.6). Dessen Methode Draw benötigt als Parameter einen Gerätekontext hdcDraw (Abbildung 9.7). In der Vergangenheit wurde an dieser Stelle in vielen Fällen grundsätzlich der Gerätekontext für das dem Steuerelement zugeordnete Bildschirmfenster übergeben. Diese unzulässige Annahme bewirkt, dass auch bei der Ausgabe auf einem Drucker die Darstellung des Steuerelements nur in der Bildschirmauflösung berechnet wird (Appleman, 1999).

```
Interface IViewObject2 : IUnknown
{
     ...
     HRESULT Draw(
          ...
          HDC hdcDraw,   // Device context on which to draw
          ...
     );
     ...
};
```

Abbildung 9.7: Interface IViewObject2 mit der Methode Draw

Solange die Steuerelemente nur wie bislang üblich auf dem Bildschirm dargestellt werden, hat dieser Fehler keine erkennbaren Auswirkungen. Dementsprechend enthält auch die Microsoft Knowledge Base keinerlei Hinweise darauf, dass dieses Problem bekannt wäre. Werden die Rich Textbox-Steuerelemente beispielsweise durch Label Controls ersetzt, tritt das Problem nicht auf. Werden die Word-Dokumente mit bestimmten Versionen des Internet Explorers als alternativem Active Document Container geöffnet, ist der Ausdruck ebenfalls einwandfrei. Da das Fehlverhalten nur vom Hersteller behoben werden kann, wird die Problematik an dieser Stelle nicht weiter verfolgt.

9.3.3 Dokumentbaustein zur Visualisierung statischer Systeme

Zur Visualisierung von statischen Systemen einschließlich der Einwirkungen und der Auflagerreaktionen sowie von Momenten- und Querkraftdiagrammen wurde ein spezieller Dokumentbaustein entwickelt. Er stellt Funktionalität zur Visualisierung von Einfeld- und Durchlaufträgern bereit.

Im Rahmen der Pilotimplementierung wird auf eine exakte Darstellung der Umhüllenden der Momenten- und Querkraftfunktion verzichtet. Stattdessen werden idealisierte Darstellungen der extremalen Schnittgrößen angezeigt, auch wenn diese aus verschiedenen, sich möglicherweise gegenseitig ausschließenden Lastfällen stammen (Abbildung 9.8). Der Implementierungsaufwand war deshalb mit 1.700 Codezeilen moderat.

Im Gegensatz zu den tabellenbasierten Dokumentbausteinen ist der Ausdruck der Dokumentbausteine zur Visualisierung mit der vollen Druckerauflösung möglich. Damit kann die prinzipielle Gangbarkeit des Weges der Integration von Steuerelementen in Standard-Textverarbeitungsdokumente demonstriert werden.

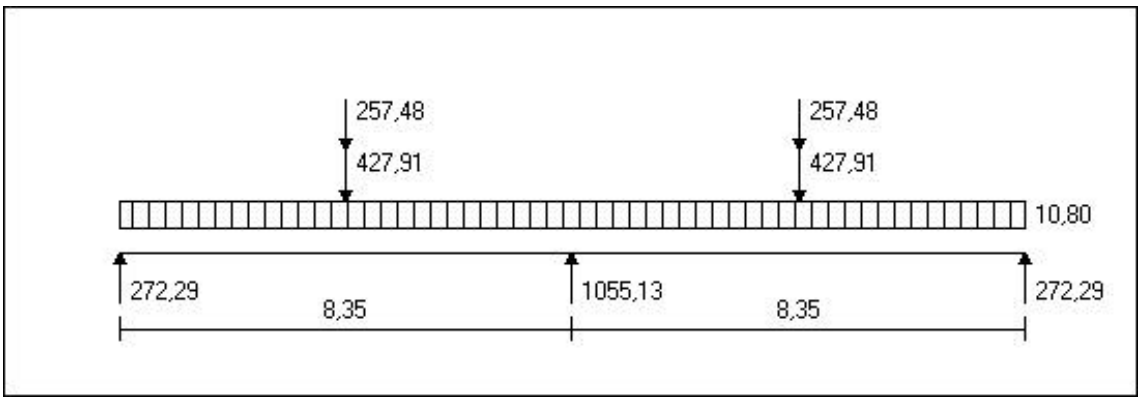


Abbildung 9.8: Dokumentbaustein zur Visualisierung von statischen Systemen

9.4 Implementierung der Bauteiltypen

Im Rahmen der Pilotimplementierung wurden zwei Dokumentvorlagen zur Bearbeitung von einachsig gespannten Vierfeldplatten und von Zweifeldträgern realisiert. Der Entwicklungsaufwand ist mit insgesamt 500 Codezeilen sehr gering. Die Komplexität von Tragwerkseditor, Dokumentbausteinen und sonstigen Software-Komponenten bleibt dem Entwickler an dieser Stelle vollkommen verborgen. Somit müssen im Wesentlichen nur unter fachlichen Gesichtspunkten Eigenschaften von Dokumentbausteinen abgefragt und anderen Dokumentbausteinen zugewiesen werden. Die Implementierung der Dokumentvorlagen gestaltet sich sehr einfach. Infolgedessen kann die Erstellung von Dokumentvorlagen auch durch informationstechnisch interessierte Tragwerksplaner erfolgen. Neben den Scripten zur Regelung des Informationsflusses sind in den Dokumentvorlagen bereits Formatvorlagen, vorgefertigter Text, Erläuterungen und Skizzen enthalten.

9.5 Implementierung der sonstigen Software-Komponenten

9.5.1 Gleichungslöser

Der Gleichungslöser stellt eine minimale Implementierung des Eliminationsverfahrens nach Gauß dar. Da für dünnbesetzte Matrizen oder Bandmatrizen keine Optimierungen vorgenommen wurden, konnte der Entwicklungsaufwand für die Pilotimplementierung auf circa 100 Codezeilen beschränkt werden.

9.5.2 Schnittgrößenermittlung von Durchlaufträgern

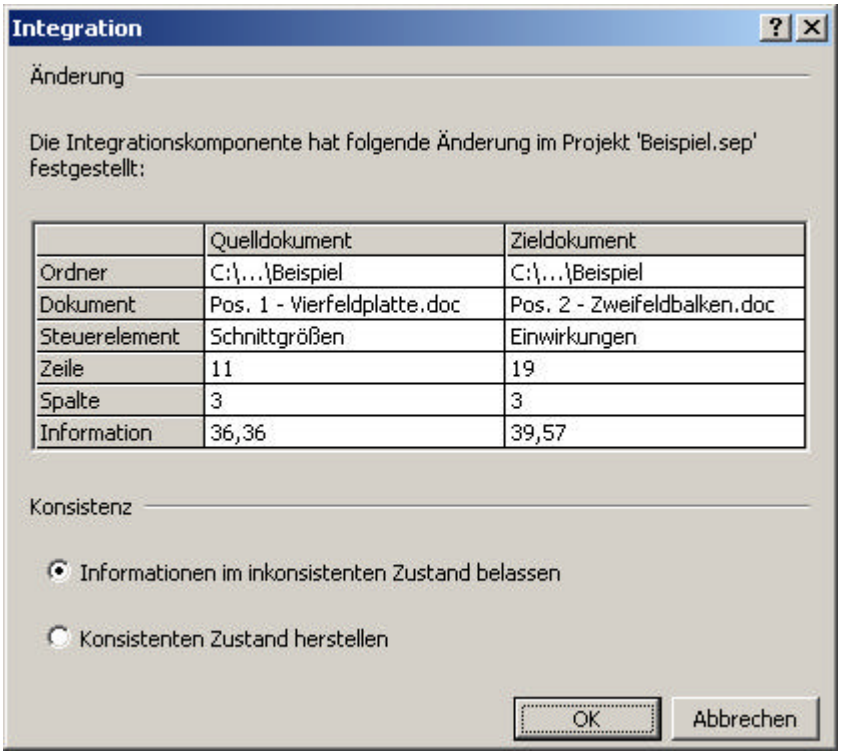
Zur Realisierung dieser Funktionalität wurden zwei eigene Software-Komponenten ohne Nutzeroberfläche entwickelt. Diese erlauben für den speziellen Fall eines Durchlaufträgers je nach dem spezifizierten Interface sowohl eine linear-elastische Berechnung als auch eine quasi-nichtlineare Berechnung durch die Umlagerung von Biegemomenten. Der Implementierungsaufwand lag bei circa 1.000 Codezeilen.

9.5.3 Integrationskomponente

Ursprünglich war vorgesehen, eine Integrationskomponente auf der Basis von Object Linking zu implementieren. Damit hätte ein einfacher Verknüpfungsmechanismus zur Verfügung gestanden, mit dem eine Integration zwischen den zu einem Projekt gehörenden Bauteilen möglich gewesen wäre. Da das dem realisierten RTF-Tabellensteuerelement zugrunde liegende Rich Text-Steuerelement OLE Linking nicht unterstützt, hätte mit verhältnismäßig großem Aufwand ein eigener Mechanismus implementiert werden müssen.

Integrationsansätze, die auf eine Interpretation der Semantik der Informationen verzichten, wurden jedoch bereits als Konzept dokumentiert und in Form von Prototypen realisiert. Da die Implementierung eines semantikfreien Integrationskonzeptes nicht den Schwerpunkt dieser Arbeit darstellt, wurde im Rahmen der Entwicklung des Prototypen auf die Realisierung einer derartigen Integrationskomponente verzichtet. Stattdessen wird auf (Bittrich, 1997), (Hinz, 1998) und (Schneider, 1999) als Vorarbeiten verwiesen.

Da die Integrationskomponente sich mit eigenen Elementen der Nutzeroberfläche in den Tragwerkseditor integrieren muss, bietet sich die Realisierung in der Form eines COM-Add-ins an. Damit würde die Integrationskomponente allen im Tragwerkseditor geöffneten Dokumenten zur Verfügung stehen. In Abbildung 9.9 ist beispielhaft eine mögliche Realisierung des Dialoges zur Anzeige von Änderungen dargestellt. Beim gegenwärtigen Entwicklungstand ist der gezeigte Dialog noch nicht mit Funktionalität hinterlegt.



	Quelldokument	Zieldokument
Ordner	C:\...\Beispiel	C:\...\Beispiel
Dokument	Pos. 1 - Vierfeldplatte.doc	Pos. 2 - Zweifeldbalken.doc
Steuerelement	Schnittgrößen	Einwirkungen
Zeile	11	19
Spalte	3	3
Information	36,36	39,57

Abbildung 9.9: Dialog der Integrationskomponente zur Anzeige von Änderungen

Eine Realisierung der Integrationskomponente sollte wie schon in (Hinz, 1998) beschrieben auf der Grundlage der Graphentheorie erfolgen. Durch einen Ablauf des Algorithmus der Tiefensuche kann verhindert werden, dass unzulässige Zyklen in den Verknüpfungsgraphen eingefügt werden. Ebenso sollte vor einer Aktualisierung der Verknüpfungen eine topologische Sortierung vorgenommen werden, so dass die Aktualisierung in einem einzigen Durchlauf erfolgen kann.

9.6 Gesamtübersicht der realisierten Pilotimplementierung

In Tabelle 9.2 sind alle Bestandteile des realisierten Prototyps einschließlich des Implementierungsaufwandes aufgeführt. Mit Ausnahme der Anwendung des Tragwerkseditors und der Dokumentvorlagen zur Realisierung der Bauteiltypen wurden alle Bestandteile als Software-Komponenten realisiert. Der Anteil der Software-Komponenten am gesamten Implementierungsumfang liegt somit bei mehr als 80%. Der Tragwerkseditor selbst kann durch das Konzept der unabhängigen Erweiterbarkeit durch externe Software-Entwickler wiederverwendet werden. Die einzigen nicht wiederverwendbaren Bestandteile sind die Bauteiltypen, die jedoch nur ungefähr 2% des Gesamtaufwandes darstellen.

Bestandteil	Komponente	Codezeilen	Anteil
Tragwerkseditor	nein	2.800	14%
Dokumentbausteine für den Massivbau	ja	13.100	66%
Dokumentbaustein zur Visualisierung statischer Systeme	ja	1.700	8%
Bauteiltypen	nein	500	2%
RTF-Tabellensteuerelement	ja	700	4%
Schnittgrößen	ja	1.000	5%
Gleichungslöser	ja	100	1%
Gesamt		**19.900**	**100%**

Tabelle 9.2: Bestandteile der Pilotimplementierung

Kapitel 10
Anwendungsbeispiel

In diesem Kapitel sollen aus der Perspektive des Tragwerksplaners die Möglichkeiten der Anwendung des Tragwerkseditors demonstriert werden. Anhand eines Beispiels aus dem Massivbau werden die einzelnen Bearbeitungsschritte veranschaulicht. Zum Abschluss werden ebenfalls unter praktischen Gesichtspunkten die Möglichkeiten der Erweiterung des Tragwerkseditors und des Informationsaustauschs mit anderen Beteiligten erläutert.

10.1 Aufgabenstellung

Als Anwendungsbeispiel zur Erprobung der Pilotimplementierung wurde das in Abbildung 10.1 dargestellte Stahlbeton-Tragwerk gewählt.

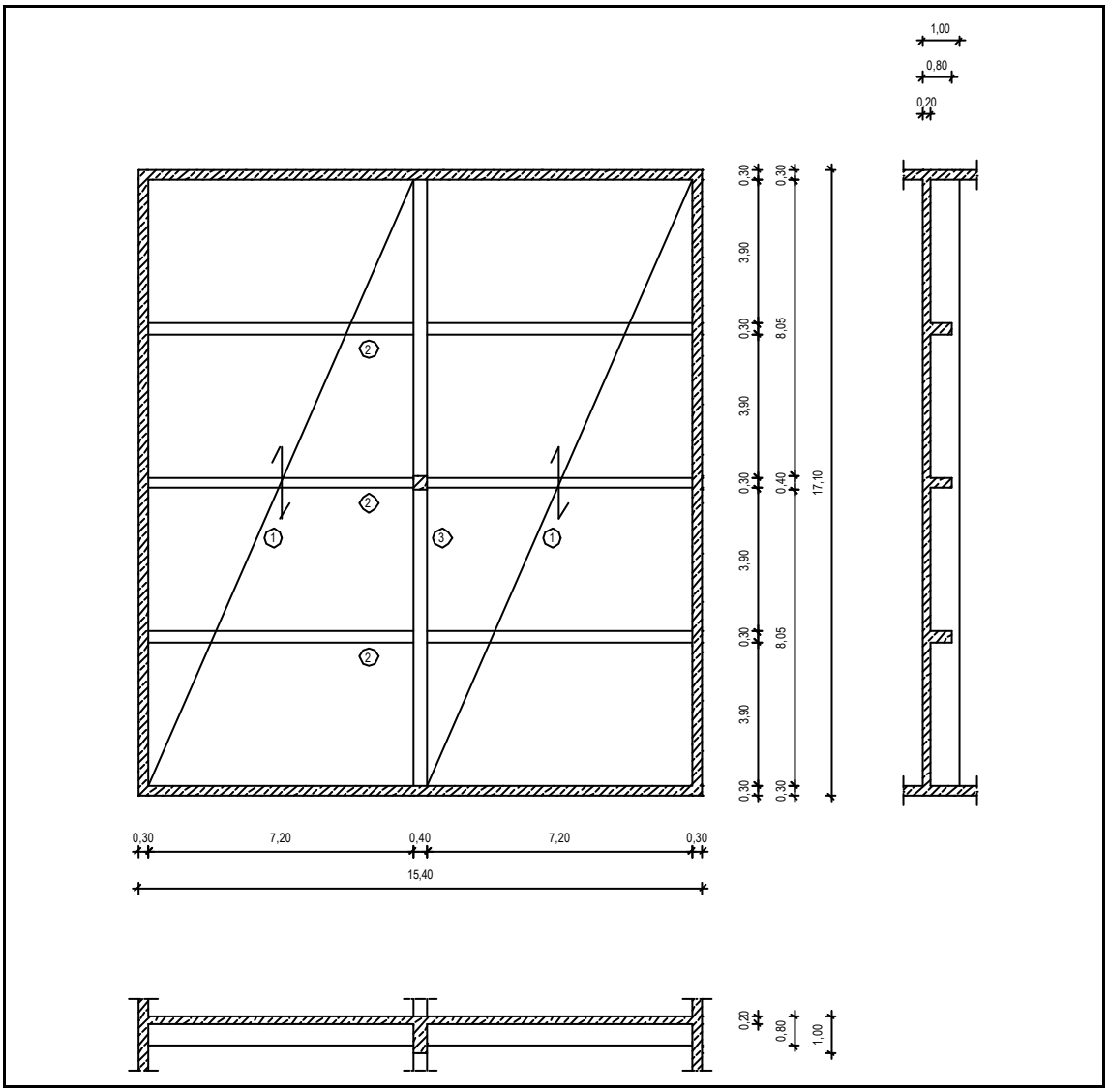


Abbildung 10.1: Positionsplan des Anwendungsbeispiels

Für die abgebildete Deckenplatte eines Bürogebäudes sind die Schnittgrößen zu ermitteln sowie die Nachweise der Tragfähigkeit und der Gebrauchstauglichkeit zu führen. Die Bearbeitung ist in einem Tragwerksbericht zu dokumentieren. Die Modellierung erfolgt mit drei Positionen als dimensionsreduzierte Teiltragwerke. Die Decke wird als einachsig gespannte Deckenplatte mit Durchlaufwirkung über vier Felder modelliert. Als Unterstützung dienen die Außenwände sowie drei über je zwei Felder durchlaufende Nebenunterzüge und ein ebenfalls über zwei Felder durchlaufender Hauptunterzug.

10.2 Arbeitsablauf

Der Anwender startet den Tragwerkseditor. Dieser enthält ein leeres Projekt, das noch keine Positionen enthält. Da der Arbeitsablauf im Wesentlichen für alle drei Positionen gleich ist, wird nur die Bearbeitung der ersten Position beschrieben. Bei der Bearbeitung der übrigen Positionen ist sinngemäß zu verfahren.

1. Wahl der Dokumentvorlage

Zur Bearbeitung erstellt der Tragwerksplaner ein neues Dokument. In der Dialogbox werden alle im Ordner Tragwerksplanung verfügbaren Dokumentvorlagen angezeigt (Abbildung 10.2). Nachdem der Anwender seine Auswahl getroffen hat, wird auf Grundlage der gewählten Vorlage ein neues Dokument erstellt. Darin sind vorgefertigte Texte, Dokumentbausteine und Scripte zu deren Verknüpfung enthalten.

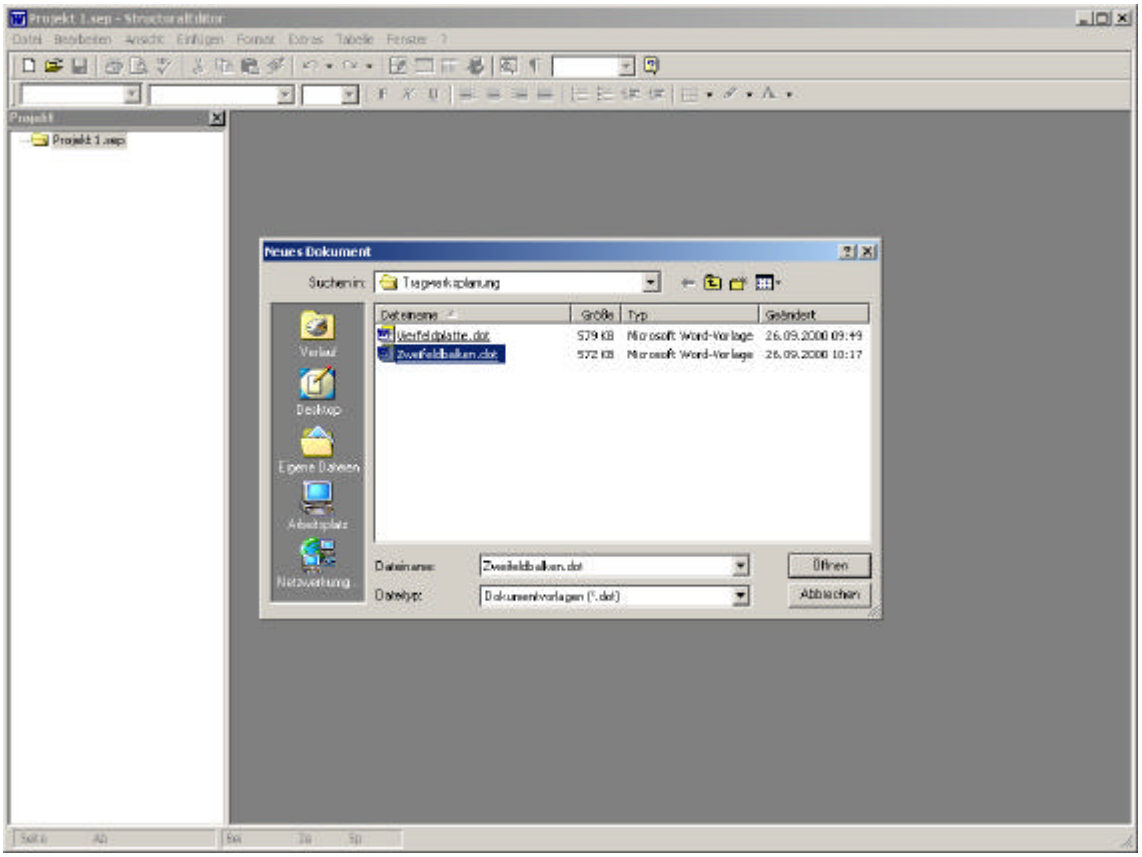


Abbildung 10.2: Auswahl der Dokumentvorlage

Bei der Erstellung des Dokuments werden alle enthaltenen Dokumentbausteine mit zweckmäßigen Vorgabewerten initialisiert. Dem Anwender obliegt es, Schritt für Schritt sämtliche Dokumentbausteine durchzugehen, die konkreten Werte zu spezifizieren und die Erfüllung der Nachweise zu überprüfen. Gegebenenfalls müssen die Dokumentbausteine iterativ bearbeitet werden.

2. Spezifikation der Geometrie

Nachdem auf der Grundlage der ausgewählten Dokumentvorlage ein Dokument erstellt worden ist, muss der Tragwerksplaner die Bauteileigenschaften festlegen. Im zweiten Schritt ist folglich die Spezifikation der Bauteilgeometrie vorgesehen. Die Eingabe erfolgt grundsätzlich über Dialoge mit Registerkarten. Über die Registerkarten erfolgt eine Trennung der Parameter des Längsschnitts (lichte Spannweite, Auflagertiefen) und des Querschnitts (Breite, Höhe, Querschnittsform und gegebenenfalls Profilierung). In einer weiteren Registerkarte müssen die geschätzten Durchmesser der Bewehrung sowie die erforderliche Betondeckung spezifiziert werden. In der letzten Registerkarte können verschiedene Optionen zur Darstellung des Dokumentsbausteins (Spaltenbreite, Darstellung des Gitters) ausgewählt werden (Abbildung 10.3).

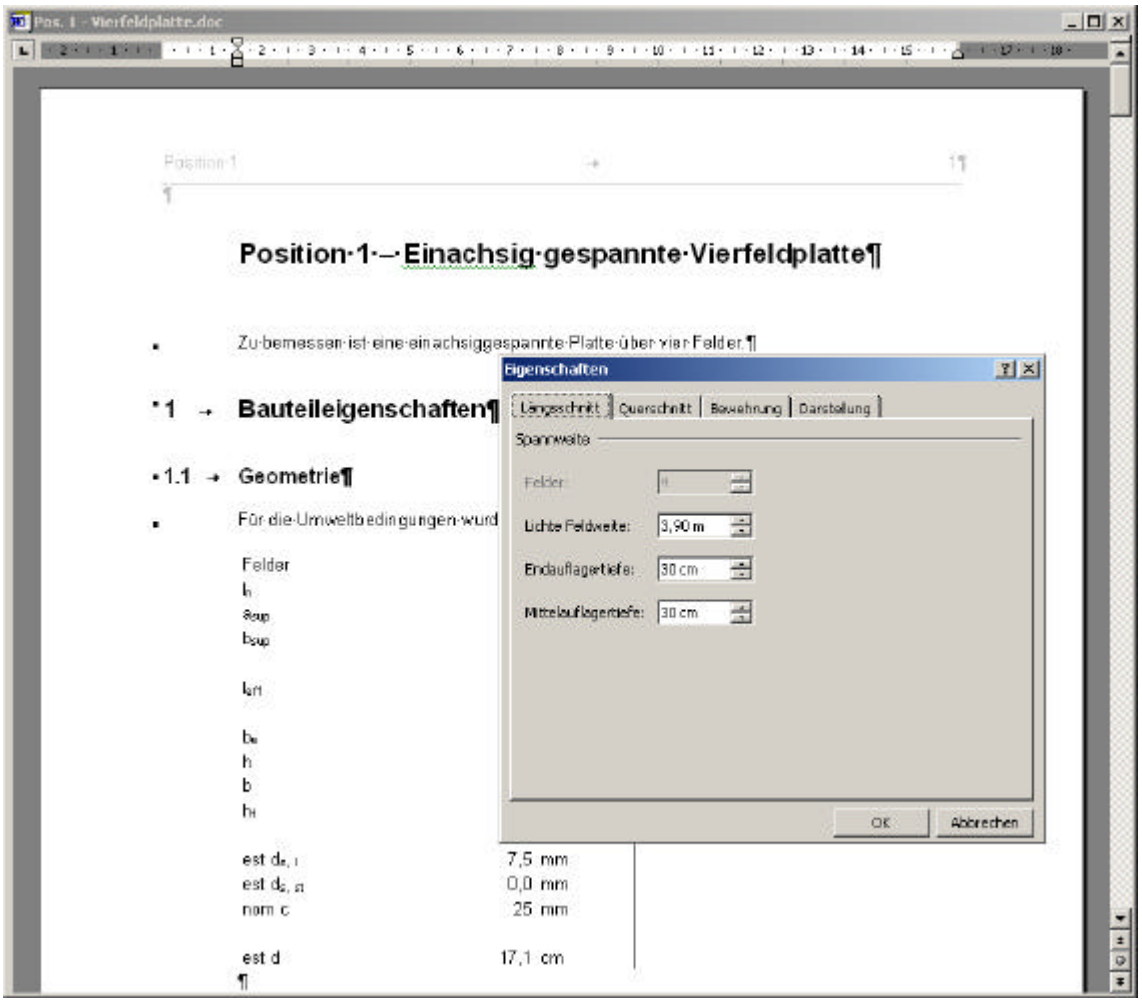


Abbildung 10.3: Dokumentbaustein und Dialog zur Spezifikation der Geometrie

Sobald der Dialog verlassen wurde, können aus den spezifizierten Informationen die effektive Spannweite und die Nutzhöhe des Querschnitts berechnet werden. Von den im Dokumentbaustein Geometrie spezifizierten Parametern und den daraus errechneten Ergebnissen sind auch andere Dokumentbausteine betroffen. Sobald die Aktualisierung eines Dokumentbausteins abgeschlossen ist, wird ein entsprechendes Ereignis ausgelöst. Im Script ist eine Ereignisbehandlungsroutine enthalten, durch welche die spezifizierten Parameter und die daraus errechneten Ergebnisse an die davon betroffenen Dokumentbausteine weitergeleitet werden. Sofern bereits alle übrigen zur Berechnung erforderlichen Parameter vorliegen, werden die davon betroffenen Dokumentbausteine automatisch aktualisiert.

3. Spezifikation der Materialeigenschaften

Als dritter Schritt erfolgt die Spezifikation der Materialeigenschaften. Der Tragwerksplaner wählt aus der Liste der verfügbaren Materialien die gewünschte Betonfestigkeitsklasse und Betonstahlsorte aus und verändert – sofern erforderlich – die vorgegebenen Teilsicherheitsbeiwerte. Aus diesen Informationen können die charakteristischen Werte und die Bemessungswerte der Materialien bestimmt werden (Abbildung 10.4).

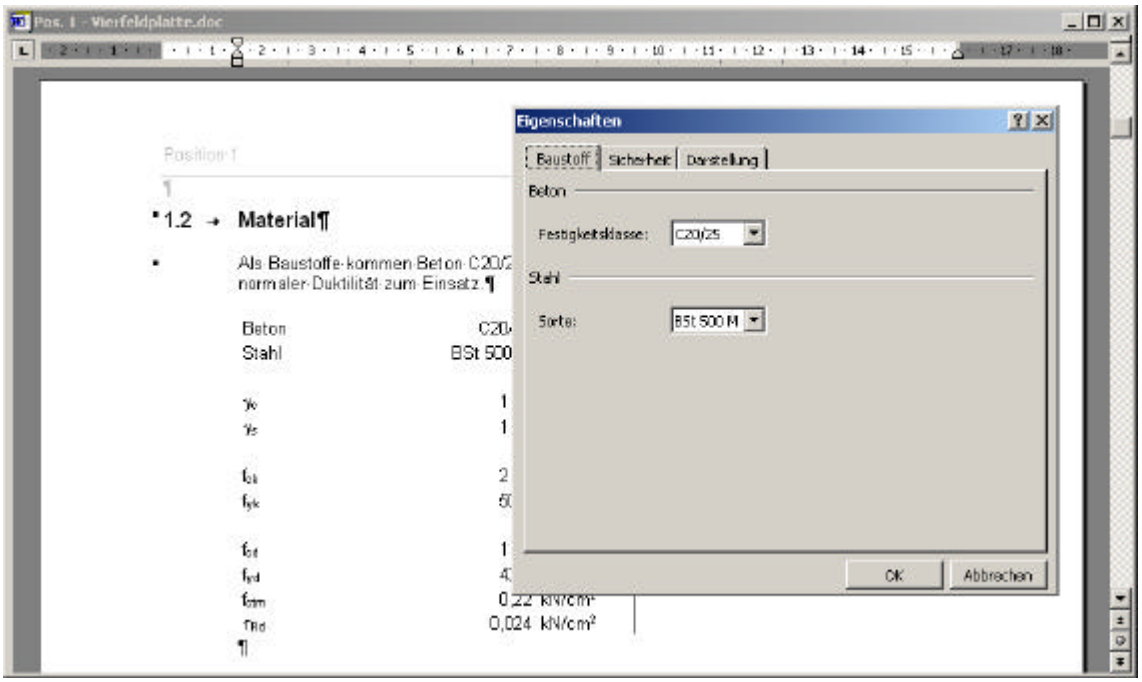


Abbildung 10.4: Dokumentbaustein und Dialog zur Spezifikation der Materialeigenschaften

4. Spezifikation der Einwirkungen

Im vierten Schritt erfolgt die Spezifikation der Einwirkungen. Das Eigengewicht der Rohbauten wird automatisch aus der Bauteilgeometrie ermittelt. Vom Tragwerksplaner müssen die ständigen und veränderlichen Lasten festgelegt werden. Außerdem müssen die entsprechenden Teilsicherheits- und Kombinationsbeiwerte ausgewählt werden (Abbildung 10.5). Aus diesen Informationen können die charakteristischen Werte und die Bemessungswerte der Einwirkungen bestimmt werden.

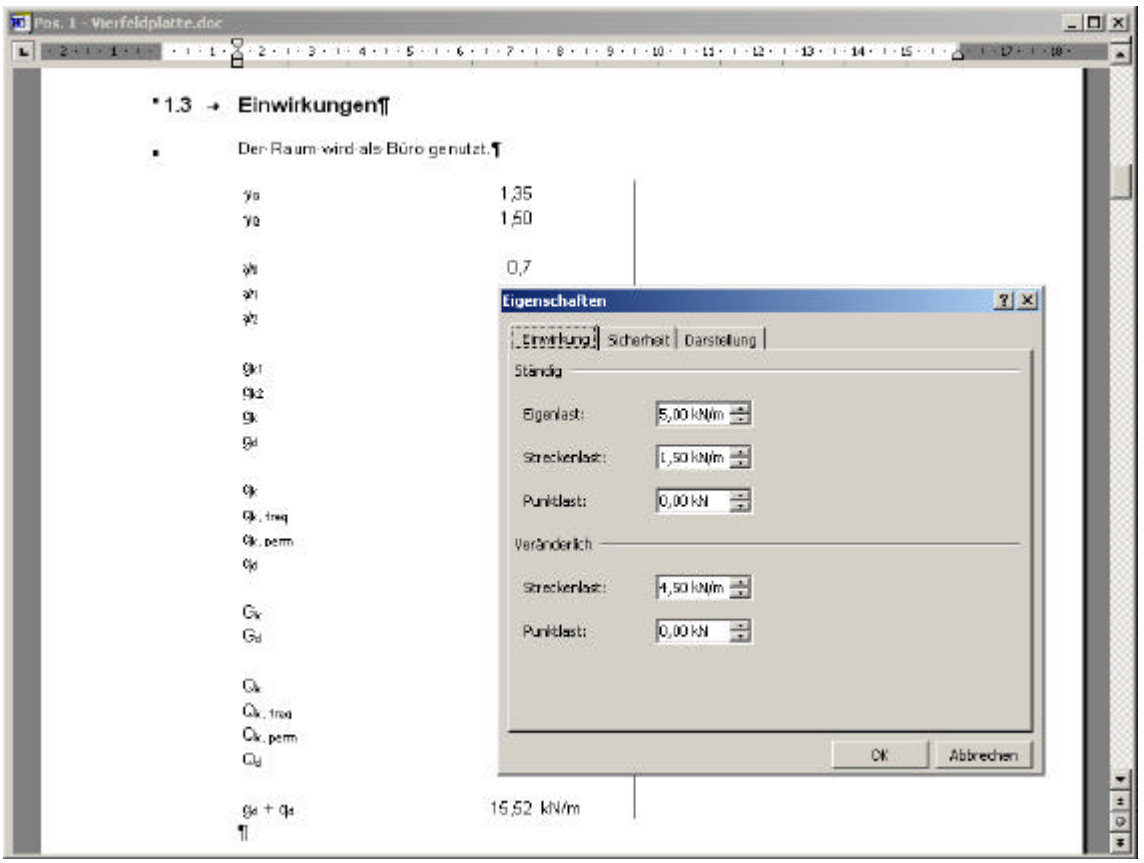


Abbildung 10.5: Dokumentbaustein und Dialog zur Spezifikation der Einwirkungen

5. Ermittlung der Schnittgrößen

Anhand der in den Schritten 2 bis 4 spezifizierten Parameter können ohne weiteren Eingriff des Anwenders die Schnittgrößen ermittelt werden. Die Ausgabe der Schnittgrößen erfolgt in tabellarischer Form in den Grenzzuständen der Tragfähigkeit und der Gebrauchstauglichkeit an den für den Bauteiltyp relevanten Stellen. Wie in Abbildung 10.6 dargestellt, hat der Tragwerksplaner die Möglichkeit, die Schnittgrößenermittlung sowohl nach der linearen Elastizitätstheorie als auch quasi-nichtlinear durch eine linear-elastische Berechnung mit automatischer Schnittgrößenumlagerung vornehmen zu lassen.

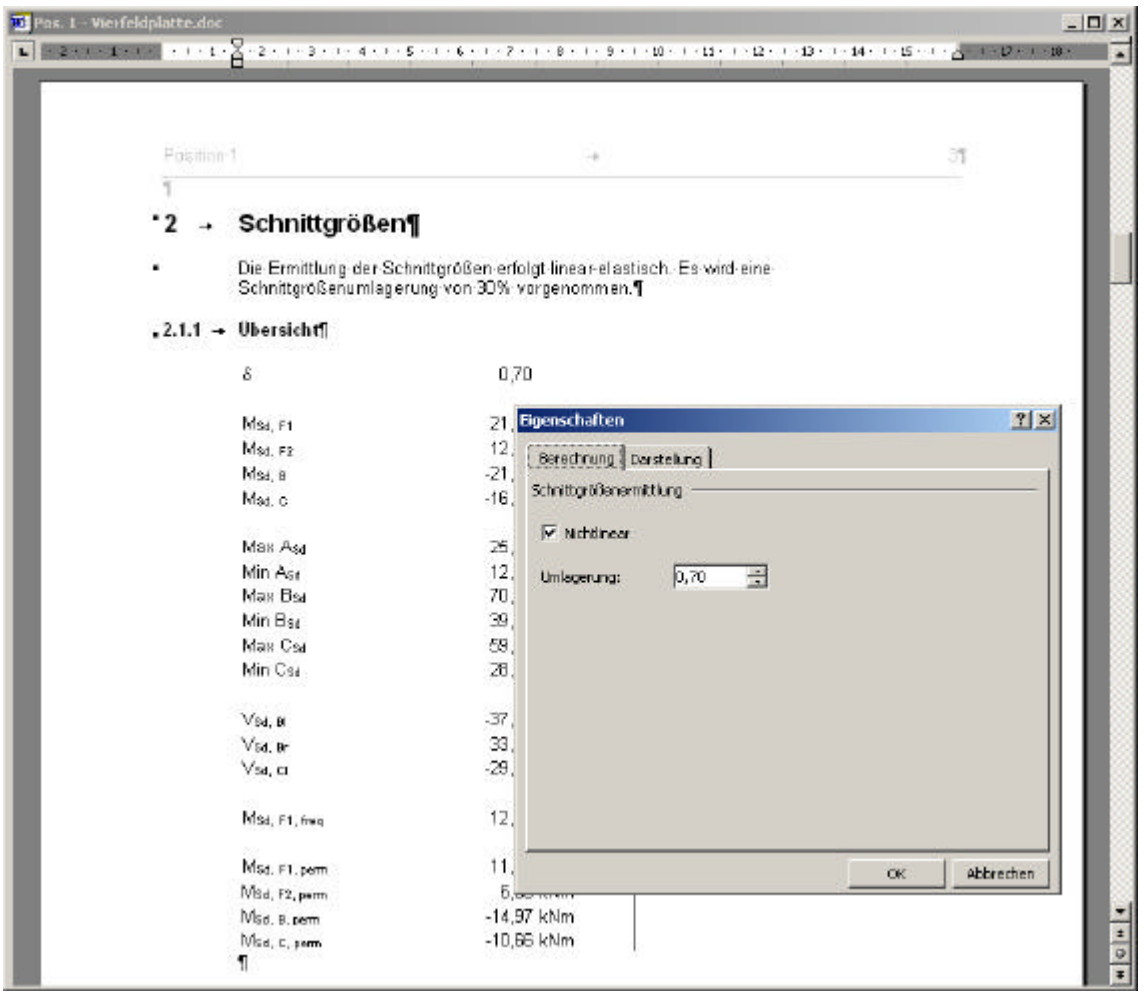


Abbildung 10.6: Dokumentbaustein und Dialog zur Ermittlung der Schnittgrößen

6. Visualisierung des statischen Systems und der Schnittgrößen

Mit den spezifizierten Informationen über die Bauteilgeometrie und den Einwirkungen sowie den im vorangegangenen Schritt ermittelten Schnittgrößen können über die entsprechenden Dokumentbausteine das statische System mit Belastung und Stützkräften sowie die Momenten- und Querkraftdiagramme dargestellt werden (Abbildung 10.7).

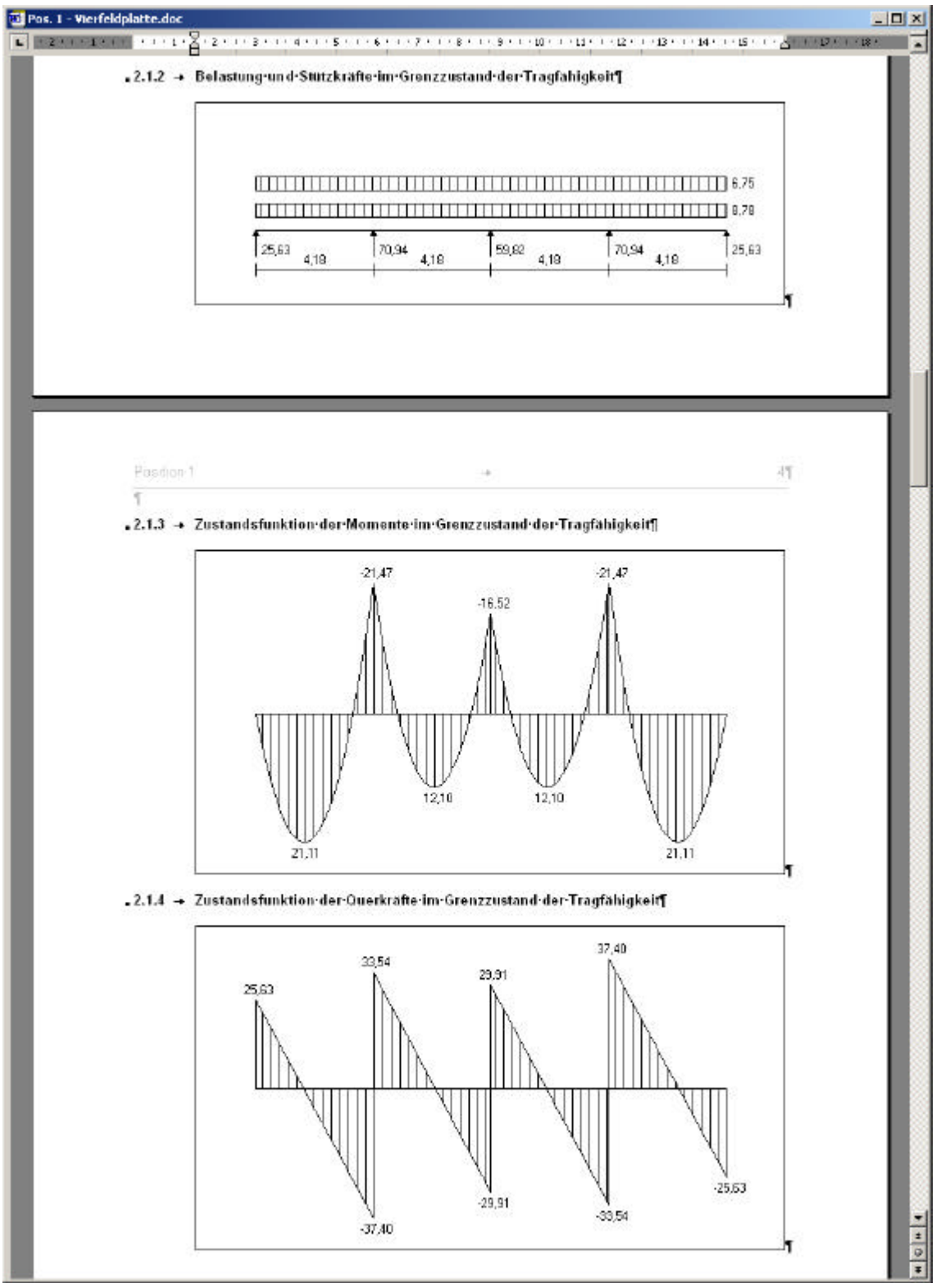


Abbildung 10.7: Dokumentbausteine zur Visualisierung des statischen Systems und der Schnittgrößen

7. Bemessung der Biegebewehrung

Nach der Ermittlung der Schnittgrößen muss zunächst im Grenzzustand der Tragfähigkeit die Biegebemessung durchgeführt werden. Die Bemessung ist sowohl in den Feldern als auch über den Auflagern, insgesamt also an vier Stellen erforderlich. An dieser Stelle wird nur exemplarisch auf die Bemessung im ersten Feld eingegangen.

Wie in Abbildung 10.8 dargestellt, ermittelt der Dokumentbaustein während der Bemessung unter anderem die Höhe der Betondruckzone. Dieser Wert wird als Kriterium zur Beurteilung der Rotationsfähigkeit des Querschnittes herangezogen und daraus die maximal zulässige Schnittgrößenumlagerung bestimmt. Gegebenenfalls wird angezeigt, dass eine Schnittgrößenumlagerung im bislang spezifizierten Umfang nicht zulässig ist. Der Tragwerksplaner muss daraufhin den Umlagerungsbeiwert reduzieren und die Schnittgrößenermittlung und die Bemessung erneut ablaufen lassen.

Sobald die Zulässigkeit der Schnittgrößenumlagerung sichergestellt ist, muss festgelegt werden, ob die Bewehrung als Stabstahl oder mit Matten erfolgen soll. Im Anschluss daran muss geprüft werden, ob die durch Durchmesser und Anzahl beziehungsweise den gewählten Mattentyp bereitgestellte Bewehrungsmenge größer ist als rechnerisch erforderlich. Gegebenenfalls kann die Bewehrungsmenge entsprechend angepasst werden. Sofern der gewählte Bewehrungsdurchmesser deutlich über dem ursprünglich angenommenen Wert liegt, ist im Dokumentbaustein Geometrie eine entsprechende Korrektur vorzunehmen.

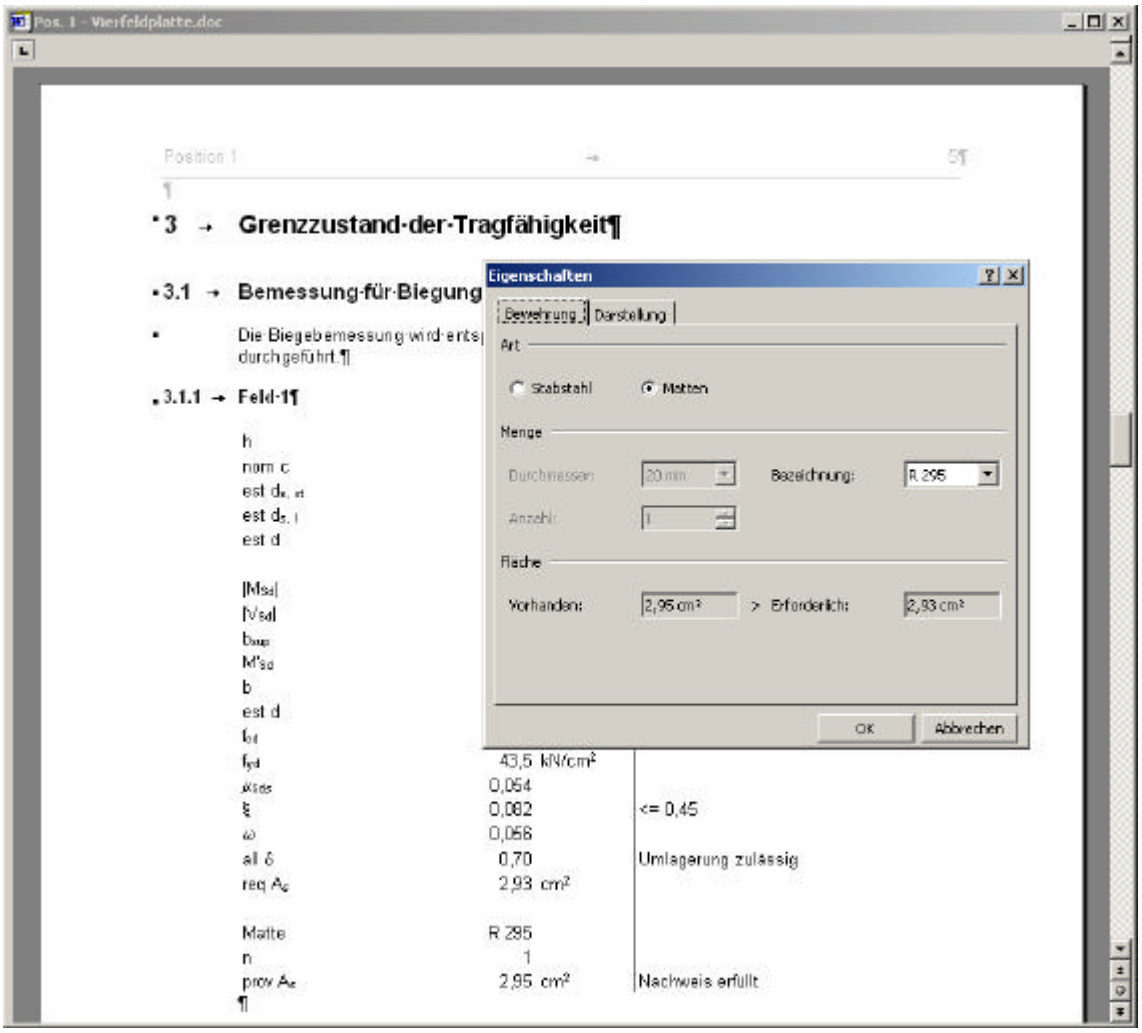
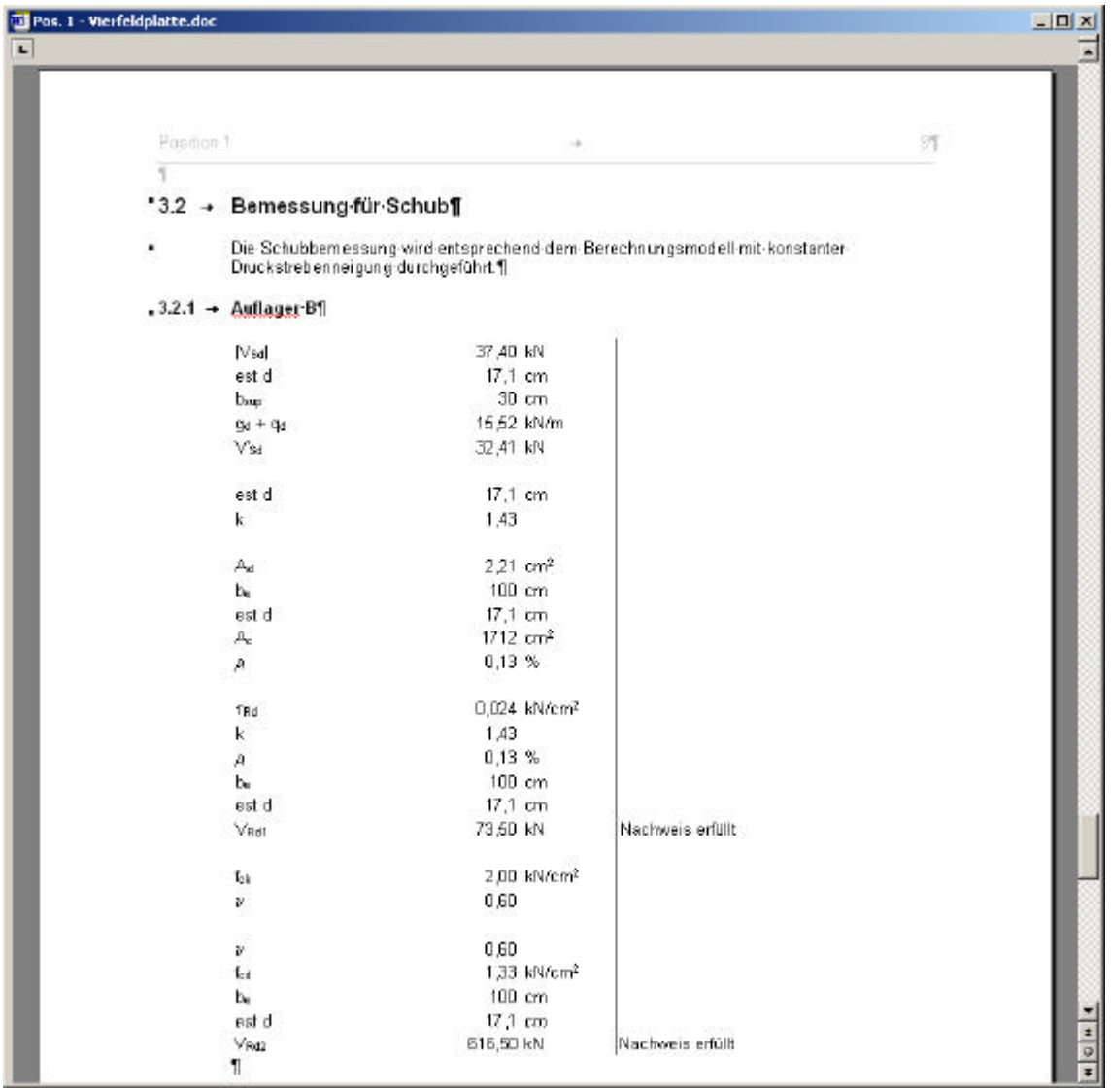


Abbildung 10.8: Dokumentbaustein und Dialog zur Bemessung der Biegebewehrung

8. Bemessung der Querkraftbewehrung

Im Grenzzustand der Tragfähigkeit ist prinzipiell neben der obligatorischen Bemessung auf Biegung auch eine Bemessung der Querkraftbewehrung durchzuführen. Die bemessungsrelevanten Stellen liegen über den Auflagern, insgesamt wäre also eine Bemessung an drei Stellen erforderlich. Da der Schubnachweis von Platten bei den im Hochbau üblichen Situationen im Regelfall ohne zusätzliche Bewehrung erfüllt ist, wird auf eine Nachweisführung an den geringer belasteten Auflagern verzichtet. Stattdessen wird stellvertretend am extremal belasteten ersten Innenauflager ein Nachweis der Aufnehmbarkeit der Querkräfte geführt. Eine ausreichende Tragfähigkeit der Zug- und Druckstreben wird separat ausgeben. Da der in Abbildung 10.9 dargestellte Dokumentbaustein von einer Erfüllung des Schubnachweises ohne zusätzliche Schubbewehrung ausgeht, ist keinerlei Nutzereingriff erforderlich.


Pos. 1 - Vierfeldplatte.doc

Position 1

3.2 Bemessung für Schub

Die Schubbemessung wird entsprechend dem Berechnungsmodell mit konstanter Druckstrebenneigung durchgeführt.

3.2.1 Auflager B

$\|V_{Sd}\|$	37,40 kN	
est d	17,1 cm	
b_{sup}	30 cm	
$g_d + q_d$	15,52 kN/m	
V_{Sd}	32,41 kN	
est d	17,1 cm	
k	1,43	
A_{sl}	2,21 cm²	
b_w	100 cm	
est d	17,1 cm	
A_c	1712 cm²	
ρ	0,13 %	
τ_{Rd}	0,024 kN/cm²	
k	1,43	
ρ	0,13 %	
b_w	100 cm	
est d	17,1 cm	
V_{Rd1}	73,50 kN	Nachweis erfüllt
f_{cd}	2,00 kN/cm²	
ν	0,60	
ν	0,60	
f_{cd}	1,33 kN/cm²	
b_w	100 cm	
est d	17,1 cm	
V_{Rd2}	816,50 kN	Nachweis erfüllt

Abbildung 10.9: Dokumentbaustein zur Bemessung der Querkraftbewehrung

9. Nachweis der Verformungen

Zum Abschluss der Tragwerksanalyse einer Position sind die Nachweise im Grenzzustand der Gebrauchstauglichkeit zu führen. Im Rahmen der Pilotimplementierung ist nur ein Nachweis der Verformungen vorgesehen. Der Nachweis wird vereinfachend über die Einhaltung von zulässigen Bauteilschlankheiten geführt. Der Nachweis läuft wie in Abbildung 10.10 gezeigt ohne Eingriff des Nutzers ab. Bei einer Nichterfüllung des Nachweises verbleibt dem Tragwerksplaner an dieser Stelle kein Gestaltungsspielraum. Folglich müsste im Dokumentbaustein Geometrie eine Vergrößerung der Bauteilhöhe vorgenommen werden und damit die Berechnung erneut durchgeführt werden.

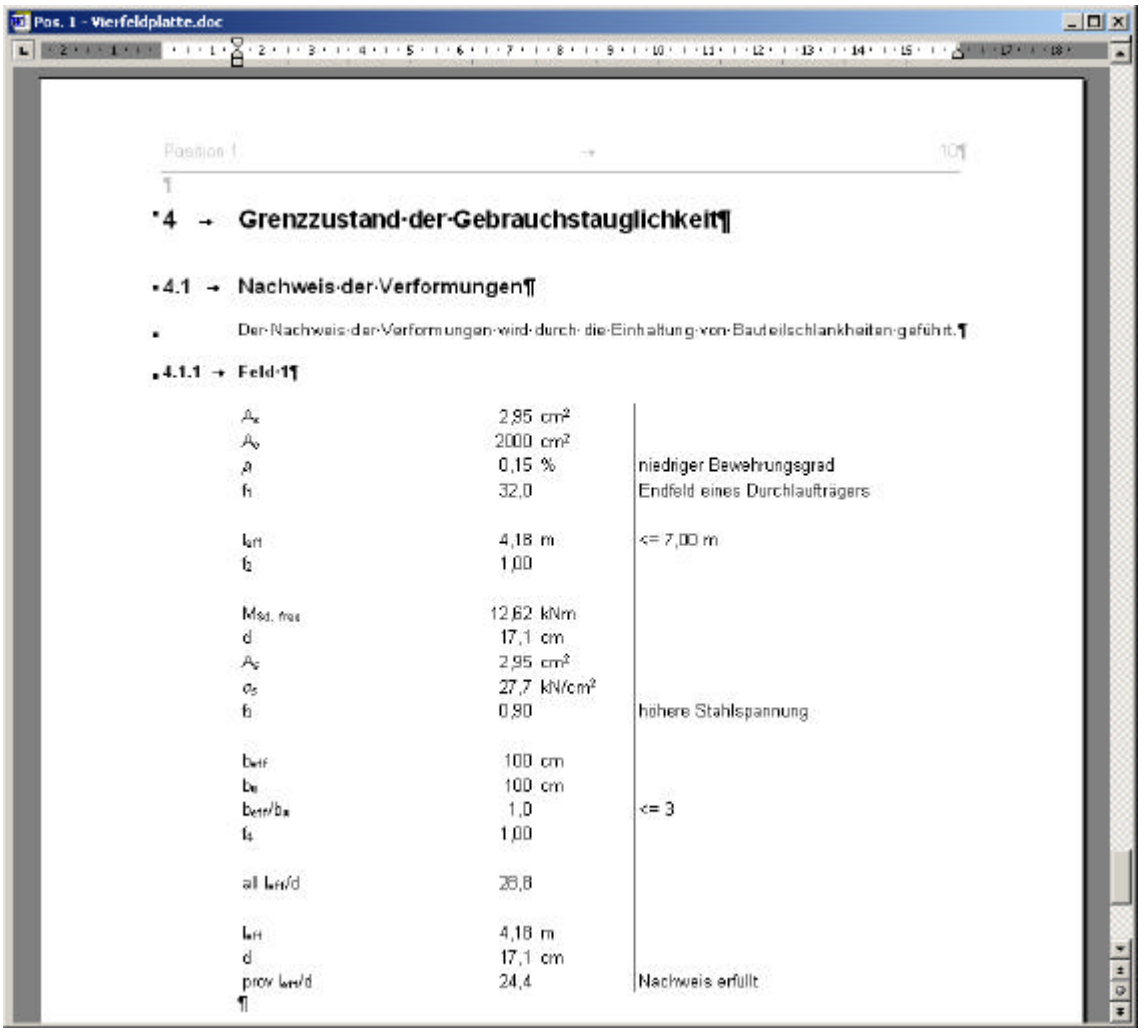


Abbildung 10.10: Dokumentbaustein zum Nachweis der Verformungen

10. Datenübernahme in nachfolgende Position

Im Anschluss an die Bearbeitung von Position 1 (Deckenplatte) sind dieselben Schritte für Position 2 (Nebenunterzug) und Position 3 (Hauptunterzug) zu wiederholen. Dazu müssen jeweils die Auflagerreaktionen der darüber liegenden Position als Einwirkungen auf die aktuell bearbeitete Position übernommen werden. Die Datenübernahme soll durch eine Integrationskomponente unterstützt werden.

10.3 Sonstiges

Neben der Vorstellung des eigentlichen Anwendungsbeispiels verdienen noch zwei weitere Aspekte Erwähnung:

- Erweiterung des Tragwerkseditors
- Informationsaustausch mit Projektbeteiligten

10.3.1 Möglichkeiten der Erweiterung des Tragwerkseditors

Im Folgenden soll die Frage der Erweiterbarkeit aus der Sicht der Tragwerksplaner anhand eines hypothetischen Szenarios diskutiert werden. Prinzipiell kann eine Erweiterung um Dokumentbausteine oder um Bauteiltypen erforderlich sein.

Infolge der vielfältigen Gestaltungsmöglichkeiten ist es sehr wahrscheinlich, dass ein Tragwerksplaner innerhalb der Dokumentvorlage einen Dokumentbaustein vermisst, beispielsweise zur Berechnung der erforderlichen Verankerungslängen der Bewehrung.

In diesem Fall muss der Dokumentbaustein neu entwickelt werden. In einer ersten Phase muss konzeptionell geklärt werden, wie die Berechnung übersichtlich in Tabellenform dargestellt werden kann, nach welchen Algorithmen die Berechnung ablaufen soll und in welcher Form die Dialogelemente gestaltet werden sollen.

Da die Grundstruktur aller Dokumentbausteine einheitlich ist, ist zweckmäßigerweise mit der Kopie eines bereits existierenden Dokumentbausteins zu beginnen, der dem benötigten Dokumentbaustein möglichst ähnlich ist. Danach sind die benötigten Eigenschaften festzulegen und die Zugriffsmethoden für die Eigenschaften zu implementieren. Weiterhin werden Routinen zum Laden und Speichern der persistenten Eigenschaften benötigt. Die genannten Schritte können stets nach dem selben Schema abgearbeitet werden. Dieser nicht fachspezifische Verwaltungsaufwand erscheint moderat und erfordert einen geringen Einarbeitungsaufwand.

Danach müssen die fachlichen Inhalte implementiert werden. Dazu gehört die Implementierung der Dialogbox, die Formatierung des Dokumentbausteins und die Implementierung der Algorithmen. Danach kann der Dokumentbaustein kompiliert und in eine Dokumentvorlage eingefügt werden. Abschließend ist auch das Script des Bauteiltyps zu aktualisieren, so dass der Dokumentbaustein in den Informationsfluss integriert wird.

Außerdem ist es wahrscheinlich, dass ein Tragwerksplaner mit dem vorhandenen Angebot an Bauteiltypen nicht auskommt und beispielsweise eine Dokumentvorlage zur Analyse eines Dreifeldträgers vermisst.

In diesem Fall ist zunächst zu prüfen, ob alle benötigten Dokumentbausteine vorhanden sind. Gegebenenfalls müssen zuerst die fehlenden Dokumentbausteine entwickelt werden. Danach empfiehlt sich wiederum, eine möglichst ähnliche vorhandene Dokumentvorlage zu modifizieren. Nicht benötigte Dokumentbausteine müssen entfernt und neue Dokumentbausteine hinzugefügt werden. Außerdem muss das Script entsprechend dem gewünschten Informationsfluss angepasst werden. Diese Anpassungen erfolgen unter rein fachlichen Gesichtpunkten.

Aufgrund der vollständigen Trennung von Tragwerkseditor, RTF-Tabellensteuerelement und der Komponenten zur Strukturanalyse einerseits und der Dokumentbausteine und Bauteiltypen andererseits ist nur ein partielles Verständnis der informationstechnischen Konzepte erforderlich. Damit können sich Bauingenieure wieder verstärkt auf die für sie interessanten fachlichen Aspekte fokussieren und zahlreiche nicht-fachspezifische Aufgaben professionellen Software-Entwicklern überlassen.

10.3.2 Informationsaustausch mit anderen Projektbeteiligten

Außerdem ergibt sich die Frage, in welcher Form der erstellte Tragwerksbericht anderen Planern oder Prüfingenieuren zugänglich gemacht werden kann, die nicht über eine Installation des Tragwerkseditors verfügen.

Der Tragwerkseditor wird nur zur effizienten Erstellung des Tragwerksberichts benötigt. Die Datenhaltung erfolgt vollständig in regulären Word-Dokumenten. Zusätzlich wird die visuelle Präsentation der Dokumentbausteine gespeichert. Das heißt, dass der erstellte Tragwerksbericht problemlos von anderen Planern oder Prüfingenieuren gelesen und mit Anmerkungen versehen werden kann, ohne dass diese über eine Installation des Tragwerkseditors verfügen müssen.

Kapitel 11
Bewertung und Zusammenfassung

Das Konzept der Verwendung von Verbunddokumenten als Nutzeroberfläche von Software für die Tragwerksplanung hat sich als leistungsfähig und aussichtsreich erwiesen. Im Folgenden wird eine Bewertung des Konzeptes aus der jeweiligen Perspektive der Tragwerksplaner und der Software-Entwickler vorgenommen. Dabei wird auf weiteren Forschungsbedarf und Möglichkeiten der Weiterentwicklung hingewiesen. Den Abschluss der Arbeit bildet ein Ausblick.

11.1 Bewertung

11.1.1 Sicht der Tragwerksplaner

Das vorgestellte Konzept für eine ingenieurgemäß gestaltete Tragwerksplanungs-Software stellt gegenüber herkömmlichen Anwendungen einen spürbaren Fortschritt dar. Der Tragwerkseditor ist so einfach und effizient zu bedienen wie Rechenblätter einer Tabellenkalkulation, bietet aber dennoch die Flexibilität und den vollen Leistungsumfang von spezieller Tragwerksplanungs-Software. Insbesondere bei der Bearbeitung der alltäglichen Aufgaben des Tragwerksplaners wird eine Steigerung von Arbeitsproduktivität und Qualität ermöglicht.

Diese Verbesserungen resultieren aus den folgenden Innovationen:

- spezialisierte Nutzeroberfläche von Tragwerksplanungs-Software
- dokumentzentrierte Nutzeroberfläche von Tragwerksplanungs-Software
- fachspezifische Erweiterung der Nutzeroberflächen von Standard-Software
- Trennung von software-technischen und fachspezifischen Aufgaben

Durch die Realisierung des Konzeptes der spezialisierten Nutzeroberfläche können Anwender zielgerichtet zur Lösung eines Problems geführt und eine optimale Anpassung an die Arbeitsabläufe der Tragwerksplanung erreicht werden. Da Spezifikation und Präsentation der Informationen mit hoher Informationsdichte und in ingenieurgemäßer Form erfolgen, wird ein effizientes und fehlerarmes Arbeiten ermöglicht.

Durch die Realisierung des Konzeptes der dokumentzentrierten Nutzeroberfläche werden die logisch zusammengehörenden Arbeitsschritte für Analyse, Bemessung und Nachweis sowie zur Dokumentation innerhalb eines Verbunddokuments zusammengefasst. Durch die gemeinsame Darstellung der Informationen in einem Dokument werden Informationsfluss und Übersichtlichkeit wesentlich verbessert. Infolge der gemeinsamen Datenhaltung können die bisher üblichen nachbearbeitungsintensiven und fehleranfälligen Import- und Exportvorgänge komplett entfallen, eine Integration ist bei diesem Konzept inhärent.

Da die Nutzeroberfläche auf der Basis von Verbunddokumenten von Standard-Software zur Textverarbeitung implementiert wurde, können die Vorteile von spezialisierter fachspezifischer Anwendungen und von Standard-Software miteinander kombiniert werden. Infolgedessen arbeitet der Tragwerksplaner von Beginn an in einer weitgehend vertrauten und damit intuitiv zu bedienenden Umgebung, so dass der Aufwand für Schulung und Einarbeitung deutlich geringer ausfällt.

Ein aus der Sicht der Tragwerksplaner besonders wichtiger Aspekt ist die weitgehend erreichte Trennung von software-technischen und fachspezifischen Aufgaben (Tabelle 11.1). Somit können software-technische Aufgaben ohne Nachteile an professionelle Software-Entwickler delegiert werden, während die Tragwerksplaner ihre Aufgaben ohne den Umweg über einen fachlich nicht versierten Entwicklers direkt bearbeiten können.

Bei einer Weiterentwicklung des Tragwerkseditors sollte der Erhöhung der Flexibilität eine besondere Bedeutung beigemessen werden. Mit der realisierten Pilotimplementierung können Stabtragwerke analysiert werden, die eine einfache Geometrie aufweisen und regelmäßigen Einwirkungen ausgesetzt sind. In der heutigen baulichen Praxis müssen jedoch beispielsweise auch polygonale Platten unter unregelmäßiger Belastung analysiert, bemessen und konstruktiv durchgebildet werden. Derartige Tragwerke werden zumeist mit Hilfe von FEM analysiert. In diesem Zusammenhang ist besonders interessant, ob auch solche Tragwerke innerhalb von Verbunddokumenten analysiert werden können. Dazu erscheint der von (Holzer, 1997), (Holzer, 1999) eingeschlagene Weg der Anwendung von auf der p-Methode basierender FEM und Trennung von Tragwerksmodell, numerischem Modell und Ergebnismodell besonders aussichtsreich.

Weiterer Forschungsbedarf besteht auch hinsichtlich der Möglichkeiten einer verbesserten Integration von Tragwerkseditor und CAD. Entwurfsunterlagen sind mittlerweile immer häufiger in Form eines dreidimensionalen Gebäudemodells vorhanden. Aus diesem Grund sollte untersucht werden, ob eine Integration von modellbasiertem CAD und dokumentenbasierter Tragwerksplanung machbar und zweckmäßig ist.

11.1.2 Sicht der Software-Entwickler

Das vorgestellte Konzept zur software-technischen Realisierung liefert aus der Sicht der Software-Industrie einen wesentlichen Beitrag, um die Entwicklung zeitgemäßer Software für die Tragwerksplanung wieder beherrschbar und rentabel zu machen. Die Implementierung konnte in äußerst effizienter Form realisiert werden. Eine Erweiterung des Tragwerkseditors durch externe Entwickler ist ebenso möglich.

Diese Verbesserungen resultieren aus den folgenden Innovationen:

- Erhöhung des Komponentenanteils durch Einsatz von Software-Komponenten ohne und mit eigener Nutzeroberfläche
- Wiederverwendung von Standard-Software durch fachspezifische Erweiterungen
- Wiederverwendbarkeit durch Realisierung eines unabhängig erweiterbaren Systems

Da neben dem bereits üblichen Einsatz von Software-Komponenten ohne eigene Nutzeroberfläche auch Software-Komponenten mit eigener Nutzeroberfläche verwendet wurden, ist der Anteil der Software-Komponenten nicht mehr auf den vergleichsweise geringen Anteil der Kernfunktionalität beschränkt. Infolgedessen konnte der durch wiederverwendbare Software-Komponenten realisierte Codeanteil deutlich ausgeweitet werden.

Dank der Wiederverwendung von durch Standard-Software bereitgestellter Funktionalität konnte der Implementierungsaufwand drastisch gesenkt werden. Außerdem ist eine kontinuierliche Weiterentwicklung der Standard-Software sichergestellt, wovon auch der Tragwerkseditor profitieren wird. Unter diesen Umständen kann eine eigene Implementierung von Funktionalität, die bereits durch Standard-Software bereitgestellt wird, wirtschaftlich nur noch in Ausnahmefällen vertreten werden.

Neben der umfassenden Wiederverwendung von intern und extern bereitgestellter Funktionalität ist durch das Konzept der unabhängigen Erweiterbarkeit auch eine Wiederverwendbarkeit durch Dritte gegeben. Externe Unternehmen erhalten dadurch die Möglichkeit, innerhalb der vordefinierten Dimensionen Erweiterungen vorzunehmen, ohne dass dazu eine zentrale Koordinierung erforderlich wäre. Dadurch können wirtschaftlich unsinnige Mehrfachentwicklungen vermieden werden.

Bereits ohne Berücksichtigung der wiederverwendeten Standard-Software konnte der Anteil der Software-Komponenten auf mehr als 80% gesteigert werden. Außerdem können Teile, die wie der Tragwerkseditor nicht in Form von Software-Komponenten vorliegen, über das Konzept der unabhängigen Erweiterbarkeit wiederverwendet werden. Der nicht wiederverwendbare Codeanteil beschränkt sich auf die Scripte der Bauteiltypen und hat mit circa 2% nur einen marginalen Anteil am Gesamtaufwand.

Es muss jedoch betont werden, dass der vorgeschlagene Ansatz nicht zu dazu führt, dass ein neues Software-System kurzfristig mit einem Bruchteil des bisherigen Aufwandes erstellt werden kann. Die Vorteile der Wiederverwendung und der unabhängigen Erweiterbarkeit kommen wegen des anfänglich erforderlichen hohen Investitionsaufwandes erst auf lange Sicht voll zur Geltung, wenn die zentrale Infrastruktur vorhanden ist und fachspezifische Software-Komponenten in ausreichender Zahl zur Verfügung stehen.

Der Aufwand für den Entwurf der allgemeinen Architektur des Gesamtsystems bleibt weiterhin hoch, da in dieser Phase weitreichende Entscheidungen, beispielsweise über die Dimensionen der unabhängigen Erweiterbarkeit getroffen werden müssen. Hingegen geht der Aufwand zur detaillierten Ausarbeitung der Architektur drastisch zurück, da ein sehr hoher Anteil der Entwurfsentscheidungen bereits implizit durch die wiederzuverwendenden Software-Komponenten vorweggenommen wird.

Besondere Sorgfalt erfordert die Evaluierung der einzusetzenden Software-Komponenten. Da die Auswahl der Software-Komponenten wesentlichen Einfluss auf das Design des Gesamtsystems hat, können diese Entscheidungen nachträglich nur mit großem Aufwand korrigiert werden. Die Auswahl dieser Software-Komponenten ist deshalb äußerst anspruchsvoll und zeitaufwändig.

Einen beträchtlichen Aufwand erfordert anfänglich die Entwicklung der Infrastruktur, wie beispielsweise des Tragwerkseditors sowie der allgemeinen Steuerelemente. Aufgrund der geringen Anzahl derartiger Bausteine, der realisierten hohen Leistungsfähigkeit und der umfassenden Wiederverwendungsmöglichkeiten ist dieser Aufwand aber langfristig gerechtfertigt. Wesentlich ist, dass Bestandteile, die in großer Zahl implementiert werden müssen und nur ein begrenztes Wiederverwendungspotential aufweisen, mit geringem Aufwand durch Entwickler mit Ingenieurausbildung erstellt werden können (Tabelle 11.1).

	Anzahl	Aufwand	Wiederver-wendung	Entwickler
Bauteiltypen (Dokumentvorlagen)	sehr groß	sehr gering	gering	Bauingenieur
Bauteiltypen (Scripte)	groß	mittel	mittel	Bauinformatiker
Dokumentbausteine (spezifisch)	groß	mittel	hoch	Bauinformatiker
Dokumentbausteine (allgemein)	gering	sehr hoch	sehr hoch	Informatiker
Tragwerkseditor	1	sehr hoch	sehr hoch	Informatiker

Tabelle 11.1: Gegenüberstellung von Entwicklungsaufwand und Wiederverwendungspotential

Beim Einsatz von Software-Komponenten steigt der Aufwand zur Qualitätssicherung und Fehlerbehebung drastisch an, da die Software-Komponenten über zahlreiche wechselseitige Kontextabhängigkeiten verfügen. In vielen Fällen kann das Debugging nicht innerhalb der Entwicklungsumgebung, sondern nur im disassemblierten Code durchgeführt werden. Selbst wenn die Ursache des von der Spezifikation abweichenden Verhaltens oder von Fehlern identifiziert werden konnten, ist ein meist erheblicher Mehraufwand zur Fehlerbehebung oder für Work-arounds erforderlich.

11.2 Zusammenfassung

Mit dieser Arbeit konnte gezeigt werden, dass Verbunddokumente als Nutzeroberfläche von Tragwerksplanungs-Software prinzipiell eingesetzt werden können und dass dieser Ansatz gegenüber herkömmlichen Anwendungen zahlreiche Vorteile bietet.

Das zentrale Anliegen von Anwendern und Software-Herstellern wird zukünftig die Verfügbarkeit beziehungsweise die Entwicklung von Tragwerksplanungs-Software mit ergonomischen Nutzeroberflächen sein. Mit der Wiedereinbeziehung der Tragwerksplaner in den Entwicklungsprozess von Tragwerksplanungs-Software steht dafür ein vielversprechender Ansatz zur Verfügung.

Es bleibt nach wie vor Aufgabe der Software-Entwicklung, mit möglichst geringem Ressourceneinsatz den Wünschen der Anwender bestmöglich zu entsprechen. Mit Wiederverwendung und unabhängiger Erweiterbarkeit stehen auch für diese Aufgabe aussichtsreiche Konzepte bereit. Die Demonstration der Umsetzbarkeit in ein kommerzielles System steht jedoch noch bevor.

Literatur

Appleman, D., 1999

"Creating scaleable/printable controls with VB an SpyWorks 6", Desaware, Campbell

Beucke, K., 1993

"Stand der Integration von Statik und Bemessung im Entwurfs- und Konstruktionsprozess", Tagung Baustatik/Baupraxis BB5, München

Beucke, K., 1996

"Stand der Integration von Statik und Bemessung im Entwurfs- und Konstruktionsprozess", Bauingenieur 2/96, Springer Verlag, Berlin

Bittrich, D., 1997

"Dokumente als Grundlage informationstechnischer Integration", Diplomarbeit, Bauhaus-Universität Weimar

Bittrich, D., 1998

"Industry Foundation Classes: Ein Überblick", Script, Bauhaus-Universität Weimar

Bittrich, D., 1999

"Verbunddokumente als Nutzeroberfläche von Software für die Tragwerksplanung", XI. Forum Bauinformatik, TU Darmstadt, VDI-Verlag, Düsseldorf

Bittrich, D., 2000

"Verbunddokumente als Nutzeroberfläche von Software für die Tragwerksplanung", Internationales Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen, Bauhaus-Universität Weimar

Booch, G., 1994

"Object-Oriented Analysis and Design", 2nd Edition, Benjamin-Cummings, Redwood City

Box, D., 1997

"Essential COM", Addison Wesley, Reading

Brettschneider, D., 1998

"Modellierung Rechnerunterstützter, kooperativer Arbeit in der Tragwerksplanung", Dissertation, Ruhr-Universität Bochum, VDI-Verlag, Düsseldorf

Brockschmidt, K., 1995

"Inside OLE, 2nd Edition, Microsoft Press, Redmond

Chappell, D., 1996
"Understanding ActiveX and OLE", Microsoft Press, Redmond

Deutsch, P., 1989
"Design Reuse and Frameworks in the Smalltalk-80 System", in Biggerstaff, T., Perlis, A., (eds.), Software Reusability, Vol. 2, ACM Press, New York

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995
"Design Patterns: Elements of Reuseable Object-Oriented Software", Addison-Wesley, Reading

Graw, G., Mester, A., 1998
"Federated Component Frameworks", Proceedings of the Third International Workshop on Component-Oriented Programming (WCOP'98), Bruxelles, Springer-Verlag

Hansen, R., Röder, J., 1996
"Objektorientierte Technologien zur Entwicklung integrierter Baustatiksoftware", VIII. Forum Bauinformatik, BTU Cottbus, VDI-Verlag, Düsseldorf

Hinz, O., 1998
"Objektorientierte Modelle und Werkzeuge für den Einsatz von Komponentensoftware in der Tragwerksplanung", Dissertation, TU München, Shaker-Verlag, Aachen

Holzer, S., 1997
"Gestaltung ingenieurgemäßer Statiksoftware", Bauingenieur 3/97, Springer Verlag, Berlin

Holzer, S., 1999
"Modellierung und Berechnung mit der p-Version der finiten Elemente", in Meskouris, K., (ed.), Baustatik-Baupraxis 7, Aachen, A. A. Balkema, Rotterdam

Huhnt, W., 1997
"Informationstechnische Integration im Bauwesen durch Nutzung fachspezifischen Anwenderwissens", Internationales Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen, Bauhaus-Universität Weimar

IBM, 1994
"The System Object Model (SOM) and the Component Object Model (COM): A comparison of technologies from a developer's perspective", IBM Corporation, Austin

Jacobson, I., 1993
"Object-Oriented Software Engineering", Addison-Wesley, Reading

Kühnemann, E., 1992
"Entwicklung eines Statikeditors – Rechnerunterstützte Statik", RIB, Stuttgart

Kühnemann, E., Burmeister, A., Bökeler, K. H., 1996
"Computergestützte Arbeitsumgebung für die Statik", Bauingenieur 2/96, Springer Verlag, Berlin

Laabs, A., 1998
"Methoden für die Modellierung mit Objekten im Bauingenieurwesen", Dissertation, TU Berlin

McIlroy, M. D., 1968
"Mass produced software components", in Naur, P., Randell, B., (eds.), Proceedings, NATO Conference on Software Engineering, Garmisch, NATO Science Comittee, Bruxelles

Microsoft, 1995a
"The Component Object Model Specification, Draft Version 0.9", Microsoft, Digital Equipment

Microsoft, 1995b
"The Windows Interface Guidelines for Software Design: An Application Guide", Microsoft Press, Redmond

Molkenthin, F., 1994a
"Modell und rechnergestützte Bearbeitung des Statischen Berichts", Dissertation, TU Berlin, Wissenschaft und Technik Verlag, Berlin

Molkenthin, F., 1994b
"Einsatz Objektorientierter Methoden im Bauwesen am Beispiel des rechnergestützten Statischen Berichts", V. Forum Bauinformatik, TU Darmstadt, VDI-Verlag, Düsseldorf

Molkenthin, F., 1995
"Modelling and Computer-Aided Application of the Structural Report", in Pahl, P. J. Werner, H., (eds.), Proceedings, International Conference on Computing in Civil and Building Engineering, Berlin, A. A. Balkema, Rotterdam

Molkenthin, F., 1996a
"Der vollständig rechnergestützte Tragwerksbericht – Forschungsgegenstand der Bauinformatik", Bauingenieur 10/96, Springer Verlag, Berlin

Molkenthin, F., 1996b
"Integration of construction, calculation and documentation – The Structural Editor", CIB-Workshop: Construction on the Information Highway, Bled

Murer, T., 1997
"The Challenge of the Global Software Process", Weck, W., Bosch, J., Szyperski, C., (eds.), Proceedings of the Second International Workshop on Component-Oriented Programming (WCOP'97), TUCS General Publications Series, No. 5, Turku

OMG, 2000
"The Common Object Request Broker: Architecture and Specification, Revision 2.4", Object Management Group

Pfister, C., 1997
"Component Software: A Case Study using BlackBox Components", Oberon Microsystems, Zürich

Röder, L., 1998a
"Eine modulare Softwarearchitektur für Berechnungen nach der Finite Elemente Methode", Diplomarbeit, Bauhaus-Universität Weimar

Röder, L., 1998b
"Komponentensoftware für Finite Elemente Berechnungen", X. Forum Bauinformatik, Bauhaus-Universität Weimar, VDI-Verlag, Düsseldorf

Rogerson, D., 1997
"Inside COM", Microsoft Press, Redmond

Rüppel, U., 1994
"Objektorientiertes Management von Produktmodellen der Tragwerksplanung", Dissertation, TU Darmstadt

Rumbaugh, J., Blaha, M., Lorenson, W., Eddy, F., Premerlani, W., 1991
"Object-Oriented Modeling and Design", Prentice Hall, Englewood Cliffs

Rumbaugh, J., Jacobson, I., Booch, G., 1997
"Unified Modeling Language Reference Manual", Addison-Wesley, Reading

Schneider, U., 1999
"Standardisierung der Kommunikation als Integrationsansatz für das Bauwesen", Dissertation, Bauhaus-Universität Weimar

Sun Mircosystems, 2000
"The Java Language Specification, 2nd Edition", Palo Alto

System Concepts, 1997
"When GUIs Fail: Usability is more than presentation", London

Szyperski, C., 1995
"Component-oriented programming: a refined variation on object-oriented programming", The Oberon Tribune, Oberon microsystems, Inc., Zürich

Szyperski, C., 1996
"Independently extensible systems – software engineering potential and challenges", Proceedings, 19th Australasian Computer Science Conference, Australian Computer Science Communications

Szyperski, C., 1998
"Component Software: Beyond Object-Oriented Programming", Addison-Wesley, Essex

Szyperski, C., Pfister, C., 1997
Workshop on Component-Oriented Programming, Summary, in Mühlhäuser, M., (ed.), Special Issues in Object-Oriented Programming – ECOOP96, Workshop Reader, dpunkt Verlag, Heidelberg

Szyperski, C., Vernik, R., 1998
"Establishing System-Wide Properties of Component-Based Systems: A Case for tiered Component Frameworks", Workshop on Compositional Software Architectures, Monterey

Udell, J., 1994
"Componentware", Byte Magazine 5/94

Weck, W., 1996
"On Document-Centered Mathematical Component Software", Dissertation, ETH Zürich

Weck, W., 1997a
"Document-Centered Computing: Compound Document Editors as User Interfaces", Turku Centre for Computer Science (TUCS) & Åbo Akademi, Turku

Weck, W., 1997b
"Independently Extensible Component Frameworks", Proceedings, International Workshop on Component-Oriented Programming, in Mühlhäuser, M., (ed.), Special Issues in Object-Oriented Programming – ECOOP96, Workshop Reader, dpunkt Verlag, Heidelberg

Werkle, H., 1997
"Untersuchung der Eignung Objektorientierter Datenbanken als Datenbasis für komplexe baustatische Berechnungen", Bericht, FH Konstanz

Werkle, H., Hansen, R., Röder, J., 1997
"Object-Oriented Databases in Software Development for Structural Analysis", Internationales Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen, Bauhaus-Universität Weimar

Werkle, H., Röder, J., Hansen, R., 1996
"Untersuchung der Eignung Objektorientierter Datenbanken als Datenbasis für komplexe baustatische Berechnungen", Bericht, FH Konstanz